EMBRACING SERENDIPITY

APPLAUFT BLOWS CISCO TELEPRESENCE OUT OF THE WATER WITH A “BLUE OCEAN” STRATEGY

SAP takes a deep dive into team soseaology & finds “Interactions are like fish. They have many scales.”

“WE DONT CARE WHAT THEY TALK ABOUT, AS LONG AS THEY ARE TALKING” - THE TAO

Finally, a video call that isn’t awkward!” - Mark Zuckerberg

Trinity College & Stanford University TEAM UP TO DESIGN

“Somebody get me some stock options!” - Mark Schar
People are profoundly short-sighted.

Despite how perpetually interconnected everyone is today online and on smart phones, the scope of our conscious awareness is still usually limited to our physical surroundings. A basic, undeniable human truth is described in the expression “out of sight, out of mind.” For us to stretch our attention any further – that is, to be mindful of something or someone that is out of sight, regardless of how large or important it is – requires deliberate effort and strong personal incentive. This problem becomes very real and very costly in the context of large-scale remote collaboration; especially for companies like SAP, which has over 50,000 employees spread out in small inter-connected teams all over the world. It is chiefly a problem of awareness.

As designers, our challenge is not to overcome this problem, but to accept it. That is the mindset we had when we designed the WaterWall. It is intended to explicitly address people’s implicit shortsightedness, and harness their existing behavior in a new productive direction. “Teaching SAP employees how to look beyond their office walls,” is hardly as powerful a goal as “helping them break down their office walls.” Rather than asking ourselves, “How might we help our users be more aware of the colleagues and resources they have beyond the ones that are obvious and salient in their daily routine?” We reframed the challenge, asking instead, “How might we transform those colleagues and resources so they are an obvious and salient part of their daily routine?” This was a major turning point. We resolved to create a catalyst in the work environment, which would make remote colleagues a salient part of one another’s local work environment.

A more explicit problem inhibiting use of remote collaboration tools is the restrictive nature of the tools themselves. Many products have tried bridging this gap between remote locations, but none have succeeded in making an effective communication tool that is both efficient and socially painless. Our initial interviews and observations at SAP confirmed that scheduling and long setup-times are often big enough deterrents to prevent people from initiating video calls. The process of starting a videocall must be simplified and streamlined. Even after it has started, though, the central interaction itself is another unsolved problem...
Social software is ubiquitous, but it’s all very direct. Here is an interesting question: What is the smallest scale of social interaction that can take place remotely? Is it being cc’d on an email? Seeing someone sign on in your buddy list? Being one person in a group phone call? Currently, there are no channels of professional remote communication that allow mutually spontaneous interactions. There is no online version of a shared glance across the room, or a casual hallway collision. This insight revealed a critical interaction gap that is currently not being filled by any existing applications: Serendipity.

The most exciting part is that these unplanned, transient exchanges are often the catalyst for new collaborative ideas. That’s why new inspirations often strike while chatting at the water cooler, rather than working at a solitary desk...

Our product was inspired out of this concept; that collaborative breakthroughs and new innovative connections will not occur unless there is first a nexus where contrasting viewpoints have an opportunity to collide. The WaterWall has a live 24/7 video-stream to ensure that no time is ever wasted on setup, and interactions can be mutually spontaneous. The “thought bubbles” display current project information that is relevant to both parties, providing a shared awareness of team progress, and some easy conversation starters. Lastly, an interactive graphic display with gestural commands gives users a carefree activity to engage in so their talks are lighter, more socially natural, and not as awkwardly direct.

The WaterWall allows remote teams to communicate in simple, transient ways during idle moments. People value indirect, micro-interactions, so it addresses them explicitly. With a constant, ambient view of the other side, it not only raises mutual awareness between distant teams, but also mutual familiarity and social ease, so that remote colleagues gradually feel comfortable sharing thoughts and problems. As a social gathering point, the WaterWall is a catalyst for new intersections between social circles, departments, and locations. It helps breaks down the walls.
Introduction

The Class

ME310 is a year-long project-based design engineering course that began at Stanford University and has been operating continuously for over forty years.

When a corporate partner engages with ME310, they receive a global student team of 6 to 8 students dedicated to their innovation challenge over a period of 8 months. Through the projects, students go through an intense and iterative process of need finding, ideation, and rapid prototyping to create and evaluate new concepts.

Company involvement provides the reality check necessary for teams to improve their innovation abilities. In the end, teams deliver functional proof-of-concept prototypes along with in-depth documentation that not only captures the essence of the designs but the learnings that led to the ideas.

Furthermore, every team in ME310 collaborates with another team from a foreign university for the duration of the project. The partnership adds diversity to the project teams and students are given the opportunity to experience true global collaboration, a skill required in today’s connected world.

The Corporate Sponsor

SAP is a world leading German software company in the field of business and financial provision. Founded in June 1972 by five former IBM engineers in Mannheim, Germany, SAP serves over 197,000 customers and provides employment to over 55,500 people worldwide (1). SAP aids companies of all sizes in a wide range of industries. It empowers these organisations and the people within them to work together in a more efficient manner.

With the help of their in-house R&D groups, in 2011 SAP introduced their leaner and more efficient development practices. SAP succeeded in becoming the first company of its size to achieve agile development cycles and utilize smaller teams.

To facilitate this leaner product cycle SAP unveiled its next generation workspace, the AppHaus, which was designed to support entrepreneurship and a start-up culture within a very large organisation. It achieved this by supporting the collaborative needs of its software developers, providing them with a flexible, adjustable working environment which better enables collaboration within project teams.

With offices in over 130 countries, remote collaboration is paramount for SAP.
Max Cougar Oswald
I hail from Santa Barbara, CA, making me the team's only U.S. citizen. Outside the d.School, you can find me playing guitar in the School of Music, or embarrassing myself in improv comedy shows. I finished my B.S. at Stanford last spring in Product Design Engineering, and am now working on a M.S. in Mechanical Engineering Design. I am skilled in the art of the Brainstorm, and was recently ranked by U.S. News as "the 3rd most deadly man in the world with a dry erase marker."

Karan Verma
I am a second year student in Management Science and Engineering specializing in Mobile and Internet Computing. I have two years of work experience at Cisco Systems as a software engineer. I hold a bachelor degree in Information Technology from the University of Delhi. I like to spend my free time reading, dancing, travelling or playing pool.
Raymond Barrett
I’m a 24 year old student doing a Master’s in Computer Science as part of the first year of a new integrated Masters course being run by Trinity College Dublin. I completed my undergraduate degree in Computer Science from Trinity College, Dublin in 2012. I have also worked in a number of different roles for feature films, one of which won the Columbine award at the Moondance International Film Festival.

Eimear Connolly
I’m a 22 year old Masters Student in the school of Mechanical and Manufacturing Engineering in Trinity College. I completed my undergraduate studies in Mechanical Engineering in 2012. During my studies I spent a summer working as a process engineer with Green Plains Renewable Energy at one of their ethanol plants in Iowa. This was a world away from where I spent my most recent summer, volunteering in Kathmandu and Kolkata.
Mark Culleton

I’m 23 and I graduated last year with a BAI in Mechanical and Manufacturing Engineering from Trinity College Dublin. I am now in my first year of a PhD in robotics, which is a continuation of work carried out in my undergraduate degree. I am a scholar of the college, which means free accommodation and free dinner daily, which of course includes a complimentary pint of Guinness! Sticking to the Irish stereotype, I also work for the Guinness Company, working as a part-time operations engineer in their heat and power plant.

Peter Weldon

I’m a 22 year old first year PhD student in Civil, Structural and Environmental Engineering in Trinity College, Dublin. For my PhD I am currently employed by a Europe-wide project entitled Green eMotion, facilitating the mass deployment of electric vehicles in the European Union. I completed my undergraduate studies in Civil, Structural and Environmental Engineering in Trinity College, Dublin in 2012, having graduated first in my class.
The Tao Jiang
I was born in Shanghai, China. I attended the University of Cincinnati, Ohio as an undergraduate. My interests include mechatronics, design, software development, swimming, Ping Pong and Japanese animations.

I am comfortable with most machine shop processes, including milling, welding, grinding and lathe. I am trilingual, speaking Chinese, English and Japanese.

Amit Badlani
Born in Mumbai and raised all over India, I grew up in a multicultural community that has enabled me to meet people from different cultures, having completely different perspectives on life. Design thinking and interactive graphics apps have really intrigued me since my freshman year.
The AppHaus is a next generation creative space, which places communication, flexibility and design at the very heart of product development at SAP. The AppHaus removes barriers to creativity by enabling users of the space to rapidly and continually reconfigure and rearrange their local environment throughout the evolution of projects.

Within SAP, the concept has been delivered to scale in Dublin. Here, Scrum teams have demonstrated the liberating impact the AppHaus has on product development, while also proving that it doesn’t sacrifice quality or focus. The Scrum format has not just been a tool for agile software development but it has also been crucial in helping employees solve the critical software bugs and problems.

The AppHaus has also being rolled out across three buildings at the SAP Palo Alto campus, a location that works closely with Dublin on next generation applications. As the concept is rolled out in locations across the SAP labs network, a new challenge emerges: “How do we improve remote collaboration in the Scrum meetings?” If the AppHaus succeeds locally through flexibility and ad hoc interactions, then how can the same sense of agility and empowerment be achieved across widely distributed locations? “How do we leverage collaboration between different geographic locations?”

Such integration will require solutions to a number of challenges:

- How can communication across locations become natural and less of an ‘event’ that must be planned?
- How can empathy and team spirit be strengthened across remote locations?
- What does flexibility and re-configurability mean in project spaces that span more than one physical location?
- How do remote teams share their diversity and perspectives to create richer collaboration?
Our team surveyed thirty randomly selected employees at SAP, Los Altos, and over half reported that they personally interact with “four people or fewer” on a daily average. SAP has an enormous workforce with myriad skills and experience in a wide variety of technical fields, but without effective means for remote collaboration, each location might as well be its own isolated entity. With the wrong tools, massive corporations like SAP can be reduced to the same local resources as small start-ups.

To be sure, small local teams do have distinct strengths of their own. In fact, many general strategies and team behaviors that are a part of “agile” software development, which has been quite successful for SAP in recent years, were originally inspired by small startups. Our team believes that these successful behaviors of small agile teams can be translated to a global scale, without sacrificing efficiency or social engagement. One of the ways Agile teams thrive is by utilizing high-frequency, low-duration meetings. It keeps everyone synchronized and privy to each others progress. Problems are revealed and solved more quickly, and tasks are broken down into subtasks to be completed in rapid succession. The term “agile” is not an acronym; these teams really are actually just moving a lot quicker than most of their competitors in the realm of software engineering, and there is no technology preventing them from having these types of high-frequency low-duration team interactions while in remote locations. Apart from certain time zones, the barriers are mostly social or behavioral.

However, this optimistic scenario we are imagining, in which Agile is a global practice, comes with a simply pair of caveats. The contingency is that both sides maintain conscious awareness of each other (i.e. loosely privy to their on-goings and availability) and that both sides have access to communication tools that enables simple and spontaneous correspondence when needed.

Unfortunately, today, neither of those things are true.
There are only two methods that people frequently use for remote collaboration at SAP. First, Cisco Telepresence meetings. These large formal meetings are well-suited for presentations and reviews, but they require exhaustive planning and create a stiflingly formal atmosphere that is not conducive to creativity or collaboration. The second method is to use ordinary webcam video chats, which have awkward spatial constraints, unreliable quality, and actually still require considerable planning and coordination on either side.

Agile software developers at SAP need to feel consciously engaged and socially at ease during collaborative remote meetings; the experience must be more immersive than a webcam, less formal than a Telepresence room, and more immediately accessible than either one.

The biggest problem with mainstream video calling tools is not necessarily a flaw in the functionality of their design, but rather in the intention of it. Ironically, the root of the problem is that these tools are designed too well for specific types of interactions (e.g. meetings, presentations, direct dialogues, etc.). By doing so, they make it exceedingly difficult to experience any other types of interaction. Ambient moments, for example, tend to be a particular challenge; without them there is nothing to fill the natural lulls in conversation, and so an awkward taint is left on the entire interaction.

Subtle as they seem, small-scale interactions like this have an important role in social dynamics. This is what we found after interviewing countless Stanford students and SAP employees about their experience with video-chatting. The dynamic of most face-to-face video chats is so incessantly direct that many people end up feeling vaguely uncomfortable whenever the conversation comes to a pause.
One bashful SAP employee named James admitted that he actually prefers lower quality video chat streams, because “then you at least have something to talk about […] and it’s not so awkward.”

“What is it about talking to the guy one desk over,” we asked ourselves, “that is so much more appealing, comfortable, and productive than talking to anyone on the other side of a video call?”

With the goal of rethinking the video call and making it a more comfortable, engaging social experience, our team did a deep dive into the world of team dynamics, and learned all about the different behaviors that exist inside healthy, productive, local teams. We learned that the most functional groups interact on wide variety of scales; ranging from face-to-face conversations (at the top), all the way down to quick glances across the room (bottom). Experiencing the whole spectrum in high frequency is what keeps groups healthy and bonded, and comfortable sharing ideas with each other. The same is true for all interpersonal relationships; breaking past the “awkward silence,” so small talk is no longer forced during ambient moments; this is often a larger milestone in the progression of a relationship than having eye-to-eye conversations is.

A Comfortable, Silent, Ambient Coffee Break
After watching many teams at SAP and in the ME310 Loft, we found that it is these small scale interactions – the subtle, ambient ones – are really what separate the close-knit teams from distant ones.

Even in a digital world, physical proximity matters. We may be perpetually connected through our devices, but that is still very different from being together. The more time colleagues spend physically together, the more ambient company they keep, the more opportunities there will be for new ideas to collide, and more collaborative breakthroughs will be generated. This is a core tenet of the ME310 design loft, which Professor Larry Leifer refers to as "a model for innovation ecosystem architecture." The ME310 Loft is designed pointedly to that every outlet for creativity is kept within 7-seconds – people nearby to share with, whiteboards to draw with, tools to make with – because that's how short the human attention span is. After our Third Eye prototype (more on that to come) we decided to make the "7-second Rule" a design requirement for our system. Whatever it might become, our remote collaboration tool must have the capability to be used in seven seconds or less.

More evidence can be seen for the value of physical proximity in the "Allen Curve" from the canonical study done at MIT. The study proved that people are four times more likely to communicate regularly with someone who is sitting 6 ft. away as they are with someone who is 60 ft. away, and only in anomalous cases do people ever interact with colleagues in separate buildings or floors.

Many technology giants in Silicon Valley have realized that collaborative environments are less about productivity and more about serendipity. They show this by branding their headquarters as a "campuses" rather than unappealing "offices." Industry leaders like Google, Apple, and Facebook are driving innovation and entrepreneurship this way, by creating fun workplaces that maximize collaboration by promoting a highly social work environment. Boundless on-sight perks are starting to replace the classic take-home ones, in an effort to motivate employees to spend as much time physically together as possible. Yahoo! has now even gone to the extreme, and actually banned their employees from working from home.
Technology Context

Many people still talk about the video call as if it is on the brink of exploding into mass market, but the technology has been available for decades now and any sort of mass adoption still remains to be seen. The hype is especially strong in the mobile phone industry. Apple used FaceTime as a central part of their iPhone 4 marketing, but after failing to spark mass adoption it has since become much less prominent in their advertising. Similarly, Microsoft purchased Skype a few years ago in hopes of boosting their tablet and phone market. Perhaps Google Glass will soon offer the missing piece, but until then, the sad fact remains: video calling has been stuck in Gartner’s “Trough of Disillusionment” for about twenty solid years.

Furthermore, the videophone as a standalone product one could purchase in a store has also failed to reach commercial fruition. At least in the case of video calling we can say that the behavior has become relatively widespread, but still only in a very narrow context: people sitting alone talking to laptops – constrained by the awkward spatial constraints, unflattering angle, and low connection quality of a tiny webcam. Although “video phones” have been big in our imagination, no hardware product that has succeeded by any means in the way that audio phones have. Video calling remains a supplemental feature, and it is because there is still something weirdly incomplete and unnatural about the social behaviors tied to video calls as we know them. A small, essential scale of interaction is still missing, and its absence peppers our video chats with subtle discomfort.
We believe that the WaterWall succeeds in filling this awkward behavioral gap. If widely adopted in industry, it could very well foster a whole new paradigm of remote communication, where always-on, ambient presence is everywhere, and all scales of interaction have the potential to be mutually spontaneous. The WaterWall could be the missing piece that is necessary to pull video calls out of the rut of “supplemental features” they are currently stuck in, and bolster them up to that next stage of the Hype Cycle, which they are long overdue for: “The Slope of Enlightenment.”
The Darling Graveyard

Every time we had to "kill our darling," as in let go of an idea we had become attached too, it earned a respectful tomb in the Darling Graveyard. The following section gives a cursory account of the most important prototypes and design concepts that we explored before arriving at our final product. Each gives a basic summary of the concept, the prototype, the tests, the findings, and the decisions it ultimately led to to make for our final design.

3. Initial Prototypes
THIRDEYE

A head-mounted webcam, equipped with a microphone and speaker for video chatting, so the user can broadcast video captured by their “third eye” to give their correspondent a more dynamic, immersive feeling of virtual presence. Most importantly, a remote viewer could follow the broadcaster into a different room, and you could effectively “share your vision” with them as you drew things out on a dry-erase board. The device also had a laser pointer to calibrate camera angle and stability. Our many prototypes were a good test for optimal viewing angle, ergonomics preferences, the viewer experience, and whiteboard usability. Employees at SAP Los Altos and SAP Palo Alto tried it and gave feedback.

SUCCESS
The ability to stand, gesture, and move around while broadcasting a video chat feels much more natural than a webcam. Viewer has a much more engaging, active, immersive role in the virtual environment.

The remote participant no longer felt “left out,” regardless of how many people on the other side. Immersion makes it more compatible with various group dynamics.

FAILURE
Shaking and “drifting” out of the focal direction is an issue. Laser helps

Users are hesitant to adopt a new piece of extraneous hardware. Sounds like it could be a hassle/burden.

Does not (immediately) alleviate concerns of setup time, which was an original goal of ours.

DECISIONS
Whatever tool, service, or system we end up designing for remote collaboration, it must be able to happen with minimal effort towards setup time, and be initiated in 7 seconds or less!

If it involves video chat, it needs to be dynamic and usable while standing, and the viewer must feel immersed in the virtual space.
Initial Prototypes

SIM SAPS
The idea is to create a virtual world of the SAP AppHaus. The virtual world would include a character for each SAP employee, which would be made to resemble the employee’s physical appearance. The virtual world provides a non intrusive insight into the working world of global teams. It facilitates collaboration, communication, and generates excitement in the workplace. Each simulated world would show the staff currently working within the office. This would be accurate with regards to location and current activity. Position would be continuously updated using a method of localization such as triangulation. The concept was mocked up and taken to SAP Dublin for feedback on the idea.

**SUCCESS**

Animation is good for alleviating people’s concerns about the level of privacy intrusion. (Not enough though...)

There is definitely a need for accurate status updates and availability.

A “peephole” has a much different connotation than a “surveillance system.” Wording matters.

**FAILURE**

People are VERY against the idea of constant surveillance. Huge privacy concern.

The real system would require very advanced and expensive human modeling/ animation/ video tracking technologies.…

Watching people secretly from your computer is “creepy.”

**DECISIONS**

A 24/7 video feed must directly address the issue of privacy, by either not directly filming work areas, or putting a substantial level of distortion/animation on the display.

Looking at a big public display rather than a private view from your computer is a more appealing option (not as creepy).
Ran into any bugs lately? Let’s show them who’s boss.

This is SAP

Pick up Excaliber (the bat) and give the pinata a whack for every bug that’s ever given you grief. Then help yourself to a chocolatey reward!
BUGSLAYER

A playful company-wide game combined with an open forum for SAP that enables one employee to raise a “bounty” for a software bug they are currently facing in their code, and another employee anywhere in the world can see the post and slay it for him.

The hope is that Blend work and play to create fun, symbiotic activities in order to catalyze collaboration between different departments and locations. We hope it will build relationships by giving people opportunities to help each other and share different skills. The physicality and tangible prizes (food and fame) will make the collaboration process more inviting and rewarding.

Conducted and filmed two days of user tests at SAP Palo Alto with a pinata, bat, poster, and an iPad with BugSlayer app. Also sent out email survey to SAP mailing list about communication habits.

SUCCESS

Over thirty responses to survey. Shocking statistics about communication! Average number of daily interactions is “4 or fewer” for over half of those who filled it out. LOCAL collaboration is in just as dire straights as remote is.

Employees liked the idea of a company wide social game to play and talk about, especially if it involved work related topics that only they would understand.

FAILURE

The immaturity and silliness of the Pinata element is NOT their style of humor. Working on other people’s bugs is solitary and so is the majority of the game...

People were very reluctant to engage with the game, or even pick up the bat. However, when we left the set up by itself, groups of 3+ were substantially more likely to try it than people in pairs or on their own.

DECISIONS

Local employees here have very little community and seldom interact outside their small social circles. Our system should create a local social gathering point that just as much improves local collaboration as it does enhance the global community.

Activities in common areas should accomodate 3+ people simultaneously, and they must be very inviting because people here are very shy.
Initial Prototypes

SCRUM STATION

1. Initial concept sketch
2. 3D model sketch
3. Prototype 1
4. Prototype 2
5. Prototype 3

CONVEX, STANDING, SCRUM TABLE

MUSICAL RITUAL
ALL GEAR ON THIS STATION!

BUY
- Music pack
- Cheap headphones
- Mic
- Side panel

Prototype sketches:
- Mushroom
- Skinny bean
- Skinny house
- Umbrella
- Fat bean
- Fat house
- Legs and footrest
SCRUMSTATION

A table that is specifically designed for quick, collaborative, group video meetings. It emulates the behaviors of a SCRUM meeting, by making everyone stand, forming an engaging huddle, and allowing everyone to see everyone simultaneously. Build functional and experiential prototypes using foam core, wood, three LCD screens, a rolling adjustable height table, a custom built table top, and a laptop hidden below. Three different prototypes were tested to determine optimal size for 3-12 people, display angle, camera angle, and the industrial / interface design’s ability to attract people from afar.

SUCCESS

Standing is decidedly better than sitting for natural communications, because people like to gesture freely, and also because they don’t want to get stuck at meetings too long, and standing encourages short meetings.

People love the idea of no setup time.

The handlebars draw people in as planned.

FAILURE

Did not succeed in sparking intrigue across the room. People commented on drab black colors.

Screens too small for panoramic spread.

Screen angle is difficult for different height variations.

DECISIONS

If we are going to use a 24/7 video feed it must be SALIENT and INVITING and INTERESTING so that people want to approach it and look at the display. Once they get closer it must engage them.

Brighter colors and bigger screens.

Make it obvious that it is intended to be used freely.
SCRUMPOLE

A “skinny transformer” version of the Scrum Station which switches between two modes for active and passive use, and doesn’t have the center table that the Scrum Station has. Various prototypes were tested to determine optimal viewing angle, ergonomics preferences, the viewer experience, and whiteboard usability. The hope is that by removing the “desk” and adding a modular folding mechanism instead, we will eliminate the problem of using too much office space. By creating schematics and a BOM, we should have an accurate understanding of manufacturing challenges, and a confident prediction of the cost/time it will require to finish it.

SUCCESS

The folded-up mode puts the LCD screens much more “on display” and makes them more noticeable in the office.

Definitely takes up less space, but it is unclear whether it is enough...

FAILURE

LOTS OF BIG COMPLICATED HARDWARE that is very difficult to model confidently without testing for material stress, structural integrity, fastener strength, pivot points, etc.

LOTS OF BIG HARDWARE.... is expensive and time consuming and probably beyond our scope. (See sketches.)

DECISIONS

This plan in infeasible for our budget, skills, and timeframe.

We are pivoting, using the same design tenets and 24/7 video stream functionality to build a concept we designed earlier in the year, which uses projections on a wall rather than LCD screens, and involves no complicated base structure.
4. Design Development
WATERWALL

We set up the first working version of the water wall with short throw projector in several locations (with a laptop being the “other side” in a nearby location) and allow random passersby to interact with it in meyer library, in the atrium, and in Huang. The hope was to get valuable user feedback on the view, the level of distortion done to the view by the animations/water/bubbles, ask them about level of social comfort, and test hand gestures to see if they are functional and/or intuitive.

SUCCESS

3 out of 10 students stopped for a second look, When hoover tower was shown on the other side nearly 8/10 stopped.
Talking/Listening through webcam was easy in library. It’s quite easy and fun to FLIRT through the waterwall. new uses :) The gestures look hilarious when viewed from the side. People giggled a lot from a distance. Works as an attractor!

FAILURE

One user confused right with left (mirror) while interacting with the gesture interface. Another user commented that only 80% of the video experience immersed in water and 20% being blank - caused confusion. Another user prefers the whole water view compared to half water view. Awkwardly cuts you off at neck line. Need all or nothing. People don’t really like the gloves...

DECISIONS

Decided to flip (bilateral) so gestures are not mirrored.
Decided to make the WHOLE display submerged in the tank, so your neck isn’t awkwardly cut off.
Also important is that the opacity on/off control is still available. So now it’s either all water or no water.
DESIGN QUESTIONS

In order to find the right answers, we first had to ask the right questions. This section highlights key interaction design questions we articulated before, during, and after testing sessions; asking ourselves these helped us make important design decisions down the line.

**HOW DO THE PRODUCT’S AFFORDANCES CHANGE AS YOU WALK TOWARDS IT?**

**WHAT IS VISIBLE AND SALIENT FROM 100 FT. AWAY, 20 FT AWAY, AND 5 FT. AWAY?**

**WHEN YOU CAN SEE PEOPLE ON THE OTHER SIDE BUT THEY DON’T ACKNOWLEDGE YOU, DO YOU FEEL IGNORED? OR DO YOU FEEL LIKE A SPY?**

**HOW DIFFICULT IS IT TO GET SOMEONE’S ATTENTION ON THE OTHER SIDE? WILL IT BE NECESSARY TO INCLUDE A “DOORBELL” THAT EMITS A SOUND OR LIGHT ON THE OTHER SIDE?**

**HOW DO THE PRODUCT’S AFFORDANCES CHANGE AS YOU WALK TOWARDS IT?**

**WHAT IS VISIBLE AND SALIENT FROM 100 FT. AWAY, 20 FT AWAY, AND 5 FT. AWAY?**

**WHEN YOU SEE PEOPLE ON THE OTHER SIDE BUT THEY DON’T ACKNOWLEDGE YOU, DO YOU FEEL IGNORED? OR DO YOU FEEL LIKE A SPY?**

**HOW ARE THE HAND GESTURES PERCEIVED OUT OF CONTEXT? HOW MIGHT A FIRST TIME VIEWER INTERPRET THEM IF HE/SHE IS APPROACHING THE SYSTEM FROM THE SIDE, SO THE DISPLAY IS NOT VISIBLE?**
Technical Hardware Description

The Water Wall is structured as a point-to-point connected system. There is an IP link transferring video and audio data in real time. The video data is processed locally frame by frame before being transferred to the remote location. Let us discuss the hardware and software aspects of the system in detail, along with the rationale behind each decision.

The hardware for the Water Well is very streamlined and cost efficient. Please refer Fig 1 for the structure diagram of the system.
Projection System

Specifications
The Water Wall uses a NEC U300X short throw projection system. It has 1024\*768 resolution with a wall mounting feature. The projector is a 3000 lumen DLP technology providing high contrast imaging.

Rationale
Water Wall is intended for a location in the office which receives heavy foot traffic: busy hallways, cafeterias and office lobbies are some potential locations. We need the projection to be large and bright enough to catch the attention of a passerby. A second requirement is that there shouldn’t be any interference between the viewer and the projection as that would lower the experience considerably. Hence the short throw nature of the projector.

We chose a projector instead of a large screen due to budget constrains. An LCD display providing similar/better experience would have costed five to six times as much.

Routing

Specifications

Rationale:
packet loss/ connection loss and a lot of connectivity issues. By setting up a dedicated link between two remote locations we are able to ensure better quality of service and fault tolerance.

Our other alternative was going through the internet instead of setting up our own Local Area Network/Wide Area Network. The Internet suffers from
**HD Camera**

**Specifications**
We use a Mini Wide-Angle HP Webcam HD 5210 with 1080p HD video quality. It supports smooth streaming through H264 compression.

**Rationale**
Ubuntu 13.04 LTS is the operating system running the software, which transfers the video feed to the remote location. We needed a webcam that would support HD video but would also be compatible with the Ubuntu operating system’s video standard. The Logitech web-camera meets the UVC video specification by Ubuntu. Secondly, the wide-angle lens allows us to capture a 120-degree wide image, which negates the limited tunnel vision of a standard webcam.

**Microphone & Speaker System**

**Specifications**
Head mounted USB microphone and standard double-speaker setup to supplement the central computer’s audio in loud environments - attached with AUX cable.

**Rationale**
A noise-canceling microphone is required to filter the ambient noise from the desired sound of people speaking with each other. The Water Wall would be placed in public locations where there would be a lot of background noise.
The Water Wall is structured as a point to point connected system. The software receives an image from the HD camera which is then sent to three modules in parallel. The first module is the face detection system which processes the image to detect faces of people who are standing in front of the Water Wall. On the bases of the position of the person, the face detection module transmits the coordinates of the person’s face to the fish animation and movement module which then instructs the fish to swim to that location and engage the person.

The second module is the gesture recognition system which tracks three gestures that help the person “browse the fish”. They are able to virtually touch the fish and thought bubbles which provide information of interest. The third module is the communication module which transmits the local unaltered image to the remote location through stream sockets. All three module execute in parallel.

On the receiving end, the communication module receives data from the remote location. The image received from the “other office” is displayed on the water wall. A Fish-tank overlay is then superimposed on the image. The fish animation and movement module executes with the default fish movements on the overlaid image unless they receive input from the face detection or gesture recognition module.

Figure 2 illustrates how the functional parts of the product all interact to create the Water Wall experience.
In this section we describe the software components in detail and rational behind each of them. Each section is backed with a need statement, user testing and feedback.

**Real Time 24x7 Audio/Video Feed**
The real time audio-video feed acts the backbone of the Water Wall. It is the core functionality of the system.

**Description**
A real time 24x7 audio-video eliminates all setup time, and provides the technical capability to support informal meetings without the need to pre-plan them.

The large scale of the projected display was chosen to be (a) near to life-size, so the experience would be similar to looking through a window into an adjacent room. and (b) to attract attention and intrigue from accross the room.

The length of 8ft. was deemed to be effective for 1-5 people at once, because that is the largest social group size we observed moving around together and clustering at the coffee pot at SAP Palo Alto.

**User Feedback**
We tested with a bunch of form factors and decided on the dimensions of the video feed on the basis of user feedback on the following questions.
By evaluating answers to these questions we found that the video feed should be comfortable for 4-5 people to view simultaneously. Fig IV illustrates the size of the screen that we initially arrived at but it turned to be inadequate. We finalized on Figure V which fulfilled our design requirements. Figure VI is another illustration of the same form factor in a different setting.

Location: The video feed should be placed in a location which receives heavy foot fall and where it is easily noticeable. A place which might be ideal for casual conversations and greetings.
Testing in the d.School Atrium
Testing to determine optimal dimensions of use
Technical Description

We transmit high quality audio and video data through streaming sockets supported by a dedicated TCP/IP link. The system is connected via a Local Area Network/ Wide Area Network depending upon the office location. (Please refer to the hardware section). A structure diagram of the transmission protocol is shown below.

Each video frame sent by the camera at the local location is sent to the frame_serve routine which connects to the network and sends it over to the remote location. The frame_get routine receives the video frame from the network and forwards it to other Water Wall modules for further processing. The same protocol is followed for Audio_out and Audio_in except there is no processing on the sound captured by the device.
Fish-tank Overlay

The fish-tank overlay is an important feature of the Water Wall from aesthetic, design and functional perspective. It solves some important problem with existing wormhole solutions.

The wormhole connecting MIT and Stanford Campuses.

II.a Need Statement

We need to address problems associated with existing 24x7 audio-video solutions. Current products create architectural dissonance in office spaces. They cause privacy concerns for office workers and may cause them to feel ‘awkward’.

II.b Benchmarking and User Interviews

We conducted detailed interviews with Stanford students and professors who are using the Wormhole placed in the Huang Building at Stanford University. We also got to talk with students from MIT who were using the Wormhole at MIT to connect with Stanford students. Our need statement is based on the feedback we received from these users. Some key feedback is included in the thought bubbles below.
there's nothing special about it, it's just glorified Skype functionality

I feel awkward talking to a person I don't know...

It looks weird, I don't know what it is

It is not interesting, it's just an empty chair each time I see it

I don't use it because I don't have any friends at MIT
II.c SOLUTION:
The Aquarium

A fish aquarium is an object of interest to most people who visit a public space. Larger the aquarium the better it is. Our solution was to superimpose a fish-tank overlay on the video feed received from the remote location. This solves all the problems faced by existing wormhole solution. The fish aquarium would encourage a passerby to stop for a moment and watch the fishes. It would remove the architectural dissonance created by the life size video feed. The first glance to the Water Wall would only convey a large aquarium. It is a careful look at the screen would reveal the true nature of the Water Wall. It will greatly reduce privacy concerns due to the reduced opacity of the video image sent to the remote side.

Getting the fish-tank to look right was an important function of our prototype. The following figure goes through the various design iterations we went through. We hit the nail on the head with our seventh attempt.

Figure. Fish-tank overlay initial sketch.
Aquarium Animation Progress

Final version that looked ‘right’
A larger challenge was to get the fishes and their movements to look real. We rendered fish movements in Maya and ran them in a loop with Pygame libraries to mimic swimming sideways, towards the glass, away from the glass, and turning motions.

Fig. Rendering Fish motions in Maya
Rendering Fish Movements

Fig. Example side to front rendering
Creating Different Fish types

- Fat blue
- Sparkling red chromium
- Yellow striped
- Purple striped
The Water Wall solves for the privacy concerns of the user in number of ways. Please refer to the block diagram that illustrates it in a step-by-step fashion.

Test Image

Image with fish-tank overlay

Bottom right: Inclusion of local feed

Overlay with rippling water effect

Ability to adjust transparency
II.d Technical Description

Overlaying the fish-tank with the video feed involves stitching two images frame by frame as they are received from the camera. The fish-tank overlay is a static unchanging image, which is added to every Pygame image received from the HD camera on the local side of the office space. Each fish in the fish-tank is an object, which contains information about its position and previous state. Based on the previous state the next image is selected which defines the shape of the fish. Playing images in a sequence helps us in creating the effect of a fish turning, moving left, moving right or facing the user. The fish object is equipped to take commands and make the fish move to particular point in the image canvas, avoid collisions and turn the fishes.

III. Face Detection, Recognition and Fish behavior

The fish-tank overlay acts as hook and encourages the user to spend an extra moment in front of the Water Wall. To really engage the user we thought using face detection and recognition module which influences the way fishes behave in the aquarium. This module makes the aquarium dynamic and responsive to its surroundings.

Note: Face recognition wasn't integrated into our final solution at EXPE due to time constraints however we discuss it here for the sake of completeness.
III.a Need Statement

People like to tap on fish-tanks because fishes respond to taps. People like feedback when they approach an object. The Water Wall we design needs to provide this feedback to engage users. Moreover, people always need an ice-breaker to engage in a conversation. At an office space, this conversation starter is often Name, Project, Skills or interests.

III.b User Interviews

Kevin Hanks, SAP

- There should be that one thing that would be the trigger for everyone to go there and use that product
- There should be some way to interact with the wormhole
- I don’t know people at MIT, that is why I don’t use the Stanford wormhole
III.c Solution

We thought that it would be great if the fish would recognize users’ presence in front of the Water Wall and swim towards their location. This would surprise the user and cause him to move two and fro to play with the fishes. At the same time using facial recognition it would be great to know the name and project of the person working on the remote location.

Fig. A person in a remote location identifies the person sitting in front of the Water Wall
Face Detection and Fish Behavior

The face detection module processes the image received by the HD web camera. The coordinates of the face are sent to the Fish Movement and Animation module which causes the fishes to swim to that location at three times the normal speed and stay there facing the user for a certain period of time. This fish behavior is clearly visible and comes as a pleasant surprise to the user. If the user turns and moves to another location the fishes follow him to that location of the Water Wall. The fish movements are calibrated so that they don’t keep moving to the vicinity of the same location after they have followed a set of coordinates once.

The face recognition although not integrated to the final EXPE product adds on as a potential feature. Once a user arrives in front of the Water Wall the system captures various photos of the user and compares them against a database of SAP employee images. If a match is found the system greets the user with their name. If a match is not found the system encourages the user to enter their name, designation and current project into the system through a simple keyboard interface. Every time the user returns in front of the water wall, the system greets them and also makes their ID available to the remote location.

III.D Technical Description

Facial detection is implemented on Python 2.7.3 platform, with usage of OpenCV image library. It could mainly be decomposed into 6 steps, listed below, for actual commented full version of code, please check appendix A-6, "Code of Facial Detection".
<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Rationale</th>
<th>Illustration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Video Capture</td>
<td>Real-time video stream caught by web-cam</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Breakdown</td>
<td>Break down video stream into series of static frames</td>
<td>Figure 5-12</td>
</tr>
<tr>
<td>3</td>
<td>Gray Scale</td>
<td>Convert image into gray format to emphasize luminance</td>
<td>Figure 5-13</td>
</tr>
<tr>
<td>4</td>
<td>Equalize Hist</td>
<td>Run equalize histogram on grayscale to increase contrast*</td>
<td>Figure 5-14</td>
</tr>
<tr>
<td>5</td>
<td>Comparison</td>
<td>Compare equalize histogram with frontal face database**</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Marking</td>
<td>Mark the detected area and then post original image</td>
<td>Figure 5-15</td>
</tr>
</tbody>
</table>

**Figure 5-12: Original Image  Figure 5-13: Gray Scale  Figure 5-14: After Histogram  Figure 5-15: Marking**

Pixel color Intensity before equalization
Equalize histogram is a method that improves the contrast in an image, in order to stretch out the intensity range. Frontal face database is a trained document of facial description encoded into xml file. The database, with the name "frontal_face haarcascade", was trained by half million frontal face pictures (positive sample) and half million pictures without frontal face (negative sample).

**Face Recognition Module**

Most face recognition module uses the Eigenfaces method for face recognition. A facial image is a point from a high-dimensional image space and a lower-dimensional representation is found, where classification becomes easy. The lower-dimensional subspace is found with Principal Component Analysis, which identifies the axes with maximum variance. In this module we use the Fisherfaces method as implemented in the OpenCV 2.4.5 documentation. The Fisherfaces method learns a class-specific transformation matrix, so the they do not capture illumination as obviously as the Eigenfaces method. The Discriminant Analysis instead finds the facial features to discriminate between the persons. It's important to mention, that the performance of the Fisherfaces heavily depends on the input data as well. Practically said: if you learn the Fisherfaces for well-illuminated pictures only and you try to recognize faces in bad-illuminated scenes, then method is likely to find the wrong components. The illumination around the Water Wall would always lie in a certain bracket and hence Fisherfaces turns out to be a good alternative.
IV. Gesture Recognition and Fish Interactions

Our goal of developing the gesture interface was to provide users a hardware-free way to interact with the Water Wall.

III.a Need Statement

A chance meeting often turns into a brainstorming session. The product needs to provide a basic scribbling functionality.

III.b User Interviews

We asked our users about what feature would make their collaboration experience a pleasure. Here’s the feedback:

As
III.c Solution

Our solution involved the following: The fishes would emit thought bubbles which represent projects that are being worked on at the remote location. The employee profiles working on those projects can be browsed by swiping a finger over that fish. The gesture infrastructure also gives users the opportunity to step into a deeper collaborative experience by providing the capability of a basic whiteboard. The users can draw and erase on the video stream using simple hand gestures.

Fig. Fishes emitting thought bubbles.
Fig. User can browse employee profiles fish by clicking on it.

Fig. Using hand gesture to access employee profiles.
IV.D Technical Description

The gesture recognition module processes the image form the HD camera. For this prototype we had the user wear a glove. We then detect the color of the glove from the image which is use to create a threshold image of the hand. Aspect ratio and maximum and minimum area of the hand is calibrated to avoid noise from objects which have similar color as that of the glove. Using OpenCV functions we calculate contours around the threshold image of the hand. A convex hull analysis on the image of the hand enables us to detect convexity defects on the hand contour. Counting these defects enables us to differentiate one gesture from another.
User Testing

Other Feedback from this user, which influenced our design:

The gestures are mirrored but the shouldn’t be. Moving right should make the pointer go right on the screen as well.

Having the fish tank cover the entire screen rather than 80% of the screen is a better experience.

I like the gesture interface, makes me feel i’m part of it.
Interaction with the fish tank is a good way to improve communication.

It is kind of a always on Skype with cool interactions.
The WaterWall is a 24/7 video connectivity solution designed to bridge the gap between remote locations and facilitate serendipitous interactions between the employees in both locations.

Core Features

1. High quality audio and video communication between remote locations, with the architectural dissonance removed through the partially overlaid semi-transparent aquarium.
2. Visually beautiful and realistic aquarium scene which is in itself attractive enough to employees to make them want to spend time in its environment – facilitating serendipitous encounters.
3. Fish behaviour which provides additional incentive to be at the screen – both through movement (e.g. swimming towards a user at the screen) and by providing useful information (which in turn can act as a seed to collaboration) to the user.

Solution

**Physical:** A large area, approximately 10’ x 6’ of a wall in each office space is used as a projection screen for a short-throw projector. On the wall, which will be visible from a large area of the office, is shown a live video stream from the remote location. Partially overlaid (covering approximately the bottom 80% of the display), with transparency, is a high quality animated virtual aquarium. On either side of this projected area are an interactive bi-directional whiteboard and a conventional projector screen, while embedded within the wall are speakers, microphones, a video camera and sensors to detect when workers are approaching the screen. In front of the screen, approximately 5-6 feet away are a small number (3-4) of high stools, on which employees can rest, interact with the screen, or simply sit with colleagues. The entire set-up would be positioned in an area that provides (both for the embedded camera and of the large screen) good visibility over as much of the office as possible, and which has high foot traffic passing behind the seating area - e.g. near a coffee machine or water cooler.

**Functional:** The fish would respond to the presence of someone at the screen by swimming towards the screen, periodically releasing speech bubbles (which would subsequently fade away) with key phrases. These phrases would be aggregated from project managers in the organisation who would tag each project (and for its duration, all team members) with specific tags about the project. The most prevalent tags would feature in the speech bubbles, with an additional feature that the sizes of the fish would be determined by the relative popularity of the top tags. Visually therefore, even at a subliminal level, the fish are indicative of where the corporate mind of SAP is focused. Additional information could be accessed through hand gestures, which would allow an interested employee to browse deeper – for example, by ‘catching a fish’ by holding their hand over it, they could...
see which employees are working (or have worked) in this particular area, which projects are relevant, what other tags might be of interest etc.

Compelling: The fish therefore serve not only as something aesthetically pleasing; they also convey rich information about the company – providing two compelling reasons for employees to spend time at the wall. During this time, serendipitous encounters between employees at both sides are facilitated – initially through the bi-directional video and audio streams, but also to deeper interaction on the projection screens and interactive whiteboards.

Optional: Users have the option to place an interactive whiteboard in the vicinity of the WonderWall. This allows the users to take their conversations straight to the ideation phase by giving them the opportunity to move straight on to collaborating on an idea. Communication can continue to occur through the WonderWall system.

Bill of Materials (i.e. Minimum Parts List)

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
<th>Shipping and Tax</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEC U300X</td>
<td>899</td>
<td>54</td>
<td>953</td>
</tr>
<tr>
<td>Microphones/Speakers</td>
<td>119</td>
<td>20</td>
<td>150</td>
</tr>
<tr>
<td>Camera</td>
<td>69</td>
<td>5</td>
<td>74</td>
</tr>
<tr>
<td>Cable Hider</td>
<td>8</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Computer</td>
<td>800</td>
<td>40</td>
<td>840</td>
</tr>
<tr>
<td>24/7 Video Feed</td>
<td>$90 - $100 per month</td>
<td>-</td>
<td>1200</td>
</tr>
<tr>
<td>Total Cost</td>
<td></td>
<td></td>
<td>3227</td>
</tr>
</tbody>
</table>

System Overview

The above figure presents the general overview of the system. Each location has a short-throw projector, camera, microphone, speakers, projection surface and a computer running the WonderWall software.
These locations are connected through the Internet or an internal network, allowing both stations to communicate with each other and transfer data (i.e., video and sound). Locally, fish represent key words chosen by team leaders or managers and intermittently release the words so that the user can differentiate between individual fish. A user can interact with the system using gesture recognition. This recognition system reads in the users hand movements and affords the user the ability to select the fish, bring up the information attached to that word and offering the option to delve deeper and get more information.

**Block Diagram of Connectivity**

![Block Diagram of Connectivity](image)

The system begins by sending its frames across the network and to the frame_get function so that it can be used as the Skype small screen feed. Frame_get also receives frames sent over the network by the remote location and passes both the remote and local frames to be used in the Animated_fish portion of the system. It is in this section where the fish tank overlay is added to the feed and gesture recognition is performed on the local frame so that the system can tell where the user is trying to interact with on the screen.
The hardware decisions for the project were mainly influenced by the discussions with the teaching team and the tests done in the design School Atrium, which was the final venue for EXPE. The key elements and the rationale for each is described below.

1. The Projection Frame
An 8 feet by 6 feet projections frame was required. The team had two choices – buy the frame of exact dimensions or borrow a 10 feet by 15 feet screen from the d.School. After some discussions with George Toye, one of the senior faculty advisors for the project and Bruce Boyd, d.School Administrative, it was clear to borrow the screen.

It is a 10 feet high and 15 feet wide huge screen that can be dismantled easily. The lower end of the frame touches the ground, therefore acting as a rigid wall. The video feed starts roughly 1 feet above the ground level and faces the Atrium wall as shown in the picture below.

The screen supported the rear projection form of presentation our team needed and it was bigger than the required dimensions.

![Screen setup in the Atrium](image-url)
The webcam is placed at the top of the video feed slightly facing downwards. The facial detection algorithm worked great with this camera position. The team also tested the glove detection algorithm and it worked for distances – 4 feet, 5 feet, 6 feet and 7 feet away from the wall. The team found out that 6 feet was sort of an optimum distance for a great experience.

The team used the money saved on screen to buy a new rear projector system. The NEC projector system supported resolutions as high as 1300*1024 to as low as 628*328. The projector also supported different aspect ratios with 3:4 being one of them. The team needed a 6 by 8 feet projection area which was achieved with maximum brightness using this projector system. The team had tested various projectors in the d.School earlier and compared each of them to finally make the decision of buying the NEC projector. However, the projector system cost about $950 with shipping which was the major cost of the project.

2. Cardboard Walls
Team SAP shared the space with Team Edeka and their food-cart was going to be placed next to our screen, covering almost the whole height (10 feet) and half of the width. So the team decided to invest least in the walls. The
teaching team helped us in borrowing some of the cardboard (four 8 feet by 4 feet sheets) pieces from the Peterson design Loft. The cardboard was covered with white colored fabric and team Edeka’s poster was displayed at the back of the wall. The team was thinking of pasting posters on it earlier but has decided against it since team Edeka’s shopping machine and poster covered more than half of the walls. The posters were displaced on the Atrium wall facing the projection frame instead.

The pictures on the next page show the exact position fastening of the cardboard walls with the 10 by 15 feet frame.

3. Space surrounding the wall
The corridor space between the two walls was filled with stools, chairs and a table containing the gloves and bluetooth headsets which were a part of the Water Wall experience.

The team wanted to distinguish the sitting area and the transition area in the alley. Kevin, a member of the Trinity College Dublin Teaching team suggested that we use carpets to demarcate the 6 feet user experience zone from the transit zone. So the team placed carpet which was 6 feet wide to separate the user experience zone. The chairs and the table were all kept on the carpet with free space on the transit area to allow people to pass by comfortably in both the directions.

![Figure 8. View of the cardboard wall.](image-url)
The part of the screen which was did not have video projected had SAP and Water Wall logos on it. The webcam was placed at the top of the video feed and users found it comfortable to look at the camera and interact with people in the other location.

The area between the cardboard walls and the frame gave us enough room to house our desktop computer and projector system for the project. There was enough space for walking around the area if anything was to be checked in the area.

Figure 9. Space between cardboard wall and screen.
6. Design Requirements

Design Requirements Review
Design requirements are elements that the team has highlighted as critical to the successful completion of the design project. These requirements must be met in order for the proposed solution to address each aspect of the design problem. The requirements focus primarily on the elements that a solution addressing the current barriers to global collaboration should contain.

In the first two quarters, a number of design requirements were compiled based on the initial brief provided by SAP and also from observations, needfinding and benchmarking. These have been assessed over the months of the project and modified if required. On top of this, as the project is becoming more focused, new design requirements have been discovered. In particular, as the group is beginning to focus in on serendipitous interactions, a number of design requirements have been introduced based on these learnings and needfinding.

Similar to the previous documentation, all the design requirements have been categorised as either given and discovered. The updated Fall design requirements are summarised in Table 0.1 below, while all newly discovered design requirements are summarised in Table 0.2.

Table 0.1: Updated Fall Design Requirements

<table>
<thead>
<tr>
<th>Given</th>
<th>Discovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflect and compliment the SAP AppHaus design</td>
<td>A strong and clean connection between remote locations</td>
</tr>
<tr>
<td>Remove the “event” nature of remote</td>
<td>Easy and accessible to users</td>
</tr>
<tr>
<td></td>
<td>Unrestricted by formality</td>
</tr>
</tbody>
</table>

Figure 0.1: Review and Development of New Design Requirements
The new design requirements were merged with the old design requirements. The combined design requirements were then divided into physical and functional requirements. They were ranked in order of importance, as determined from the user centred ME310 design procedure. For each section, there were a number of associated constraints, assumptions and opportunities. These sections are shown below.

### Table 0.2: Newly Discovered Design Requirements

<table>
<thead>
<tr>
<th>Discovered</th>
<th>Given</th>
</tr>
</thead>
<tbody>
<tr>
<td>The solution is acceptable to its intended users</td>
<td>collaboration</td>
</tr>
<tr>
<td>communicate visually A means to</td>
<td>Remove barriers to remote team creativity</td>
</tr>
<tr>
<td>Overcome cultural differences</td>
<td>Mitigate current restrictions of remote collaboration</td>
</tr>
<tr>
<td>Real-time collaboration through an interactive medium</td>
<td>Encourage more frequent remote collaboration</td>
</tr>
<tr>
<td>Provide a social side to global collaboration</td>
<td></td>
</tr>
<tr>
<td>Must be intuitive to use</td>
<td></td>
</tr>
<tr>
<td>Take what collaboration methods work locally and bring it to a global scale</td>
<td></td>
</tr>
</tbody>
</table>

The new design requirements were merged with the old design requirements. The combined design requirements were then divided into physical and functional requirements. They were ranked in order of importance, as determined from the user centred ME310 design procedure. For each section, there were a number of associated constraints, assumptions and opportunities. These sections are shown below.

### Figure 0.2: Hierarchy of Design Requirements
An individual metric and rationale was created for each physical and functional requirement. These are very important in order to help quantify each requirement and justify its inclusion in the design solution. The metric assigns a numerical value to the requirement, which is of benefit as it provides a clear target which the solution must meet in order for it to be deemed successful. While some of these metrics were created based on a group discussion, others were created using some of the estimation methods taught during the ME310 course.

By using the estimation methods, a larger problem is decomposed into more manageable problems. By doing this, the estimations become easier as each problem is now smaller and more specific. As well as this, decomposing the problem is beneficial to the final estimation, as combining smaller estimations results in errors being cancelled out e.g. an overestimation for one problem is cancelled out by an underestimation of another problem. The estimation breakdown also allows for individual elements of the estimate to be updated to more accurate values as more information comes to light.

Giving a specific example, the requirement for an easily accessible solution has the following metric; “Connection has been made and communication has begun within 7 minutes of the intention to collaborate.” This numeric value was calculated as follows.

<table>
<thead>
<tr>
<th>Table 0.3: Time Estimation to Initiate Skype Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subdivisions</td>
</tr>
<tr>
<td>Time taken to write an instant message</td>
</tr>
<tr>
<td>Average time taken for response to text message</td>
</tr>
<tr>
<td>Time to start Skype and initiate a video call</td>
</tr>
<tr>
<td>Total Time</td>
</tr>
</tbody>
</table>

Physical Design Requirements

Physical requirements centre on the tangible and ergonomic features of the solution that would encourage remote collaboration between teams and individuals. Physical requirements often encompass what the solution should be or include and are often described using nouns. Each of these requirements, their chosen metric and the rationale behind the requirement choice are tabulated below. This section also demonstrates how the WaterWall fulfils these design requirements.

<table>
<thead>
<tr>
<th>Table 3.1: Physical Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
</tr>
<tr>
<td>A strong and clean connection between remote locations</td>
</tr>
<tr>
<td>Requirements</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>SAP have 1 global network that performs as if it is a private local network. This enables streaming between two locations. The program developed by the team will make use of this network for a more seamless audio and video stream.</td>
</tr>
<tr>
<td><strong>A means to communicate visually</strong></td>
</tr>
<tr>
<td>The main feature of the system is a live, always on, video feed between two locations which allows remote employees to visually communicate effectively. Body language and facial expressions are all captured. The presence of a remote shared whiteboard system also lends itself to conveying an idea visually.</td>
</tr>
<tr>
<td><strong>Acts as a reminder of remote teams presence</strong></td>
</tr>
<tr>
<td>The video feed acts like a window into a remote office. The large screen size and the dynamic fish overlay mean that the solution will attract attention daily, and it will serve as a reminder to AppHaus employees that they can use the system to collaborate remotely with experts in specific areas.</td>
</tr>
<tr>
<td><strong>Ergonomically Designed</strong></td>
</tr>
<tr>
<td>The large screen size and the camera positions allow for people on the other side of the screen to appear life sized so it people feel comfortable using the system to have a conversation.</td>
</tr>
<tr>
<td><strong>Unrestricted by formality</strong></td>
</tr>
<tr>
<td>The fish tank layer removes formality from the system and gives it a fun feeling. As the solution is designed to foster serendipitous interactions it is geared towards casual interactions. There is no need to reserve a slot; if you see someone on the other side you can simply begin a conversation. The solution opens up the world of remote collaboration to all levels of SAP employees, not just management.</td>
</tr>
</tbody>
</table>
The solution is acceptable to its intended users

Over 70% of our target users feel comfortable and uninhibited using the solution

User perception is a barrier to use of any proposed solution

From user testing in the SAP Dublin AppHaus employees are positive about the solution and wouldn’t have a problem with installing it in their work space.

Easy and accessible to users

People feel comfortable conversing at the fish tank, and it can be used to communicate immediately if someone is on the other side

The solution will not be utilised if it is difficult to initiate and set up

The solution is always on and it has no setup time.

**Functional Design Requirements**

Functional requirements represent those which the solution much do or accomplish. This can either be intrinsic i.e. something the solution must do on its own, or extrinsic i.e. something the solution must do in interaction with its environment. As a simple analogy, functional requirements are often contain a verb. From the SAP design brief, the functional requirements should encourage a solution that would reflect and compliment SAP’s AppHaus office design.

Underneath each requirement it is explained how the WaterWall design meets the design requirements.

**Table 2: Functional Requirements**

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Metric</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encourage more frequent remote collaboration</td>
<td>A 20% increase in global collaboration within SAP</td>
<td>Solution must increase the number of instances of global collaboration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The system caters for serendipitous interactions, which will lead to more frequent incidences of remote collaboration. The shared remote white board feeds into this and ideas can be shared using it. The wordle feature also links employees to remote experts in the field they are working in, this allows them to remotely collaborate more easily.</td>
</tr>
<tr>
<td>Reflect and compliment the SAP AppHaus design</td>
<td>Collaboration between AppHaus’ increases by 20% and global serendipitous interactions occur daily.</td>
<td>To expand the flexibility and success of the AppHaus into global collaboration</td>
</tr>
<tr>
<td>Requirements</td>
<td>Metrics</td>
<td>Rationale</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>The remote shared whiteboard taps into what is currently successful about the AppHaus design, and the fish tank overlay fits into the AppHaus surroundings.</td>
<td>Over 70% of users feel that they can as easily collaborate remotely</td>
<td>By not collaborating globally, ideas may not develop to their full potential</td>
</tr>
<tr>
<td><strong>Mitigate current restrictions of remote collaboration</strong></td>
<td>Serendipitous interactions are made possible globally, and the maximum scheduling needed is an email.</td>
<td>Scheduled events are quite rigid, therefore continuous collaboration is desired</td>
</tr>
<tr>
<td>The system is always on, has no setup time and doesn’t require booking in advance. It allows for casual interactions to happen remotely, and provides a seamless way for remote employees to meet and communicate and collaborate.</td>
<td>At least one local collaboration method is adopted in a global SAP project</td>
<td>Use currently proven methods to improve global collaboration</td>
</tr>
<tr>
<td><strong>Remove the “event” nature of current remote collaboration</strong></td>
<td>Solution can be used for social interactions</td>
<td>Collaboration can be increased by developing relationships within the group</td>
</tr>
<tr>
<td>As the system focuses on allowing remote serendipitous interactions to take place it differs totally from current Remote collaboration tools on the market. It doesn’t require booking, rather it creates an opportunity for remote SAP employees to bump into each other and spark casual conversation. The wordle feature provides a context for this conversation if required.</td>
<td>Idea can be captured and conveyed to a remote team member within 3 minutes</td>
<td>To encourage global serendipitous idea sharing and increase the efficiency of the global team</td>
</tr>
<tr>
<td><strong>Take what collaboration methods works locally and bring it to a global scale</strong></td>
<td>First time user can understand how to use the device immediately</td>
<td>Necessary to ensure that the solution is adopted and used</td>
</tr>
<tr>
<td><strong>Provide a social side to global collaboration</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The solution enables people to have remote casual social interactions by striking up a conversation at the fish tank.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Remove barriers to remote team creativity</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The solution caters for remote serendipitous interactions to take place. Typically when these interactions occur, they encourage the creation of new ideas that wouldn’t have been previously considered by an employee working along. The shared whiteboard surface adjacent to the solution allows remote team members to share ideas visually.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Must be intuitive to use</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The system is always on, and has no set up time which discourages use of other similar collaboration tools on the market.

Table 3: Material Requirements

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Metrics</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 feet by 6 feet short throw rear projection system</td>
<td>To project a live video feed while allowing users to interact with the wall.</td>
<td>The device must have a high resolution with optimum brightness at 8 feet by 6 feet projection size.</td>
</tr>
<tr>
<td>Frame for surface projection</td>
<td>A Frame bigger than 8 feet by 6 feet in dimensions where the live video feed and fish tank can be projected.</td>
<td>To mock--up the corridor spaces in SAP and bring the experience of the “Water Wall” in the corridor spaces.</td>
</tr>
<tr>
<td>Wide Angle Web Cameras</td>
<td>Reliable web cameras with angular view greater than 120 degree.</td>
<td>Employees need the entire view of the room to get a good picture of the remote location. Standard 100 degree angle cameras provide a restricted view</td>
</tr>
<tr>
<td>Noise Cancellation Microphones</td>
<td>The microphone should be able to filter out upto 90 dB of noise in the common hallway/corridor space.</td>
<td>There is a lot of noise in the corridors and the microphones should not pick that up as valid data.</td>
</tr>
<tr>
<td>Nitrile (non-- latex) Blue Gloves</td>
<td>Gloves that do not stick to the user while wearing.</td>
<td>The algorithm detects the blue color of the glove. People interact with the system</td>
</tr>
<tr>
<td>Requirements</td>
<td>Metrics</td>
<td>Rationale</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>The desktop box</td>
<td>A Linux machine that can run the Open CV 2 libraries.</td>
<td>A stand-alone system connected to the internet so that it can reliably run all the processing is done on the Linux desktop box.</td>
</tr>
<tr>
<td><strong>Wireless Modem</strong></td>
<td>100 Mbps modem for effective video transmission over two locations.</td>
<td>A reliable video feed transmission between two locations over a wireless network is sometimes challenging. So to ensure the highest quality transmission, there needs to be a dedicated network setup for the video transmission.</td>
</tr>
</tbody>
</table>
7. Project Management

Split on either side of the Atlantic and consisting of five different nationalities, the team has been directly experience the global collaboration issues highlighted in the project brief. Therefore project management and effective team communication continued to play a vital role in advancing our needfinding. This section summarises the key project management aspects of the project in the Winter Semester.

Global Communication

At the beginning of the Semester, the team met using the Google Hangout platform to conduct an in-depth review of our communication and collaboration techniques from the Fall and Winter Semesters. It was agreed that while the communication experienced in the previous semester was adequate, there were ways that the team could improve this communication. The team has continually strived to maintain open and regular contact, and with this goal the team decided to improve the communication protocol in the following ways:

- It was decided to continue to hold the weekly Google Hangout meetings on a more regular basis, increasing from once to twice per week.
- E-mail contact would be far more regular in the Winter Semester, with responses expected from a member of each global team no more than 24 hours after the original e-mail had been sent.
- The team agreed to update the blog on the website created for the project (applauft.com) on a regular basis with any findings from any aspect of the project. For the Fall Semester, the website was largely redundant, and the team addressed this, highlighting it as the most viable information sharing platform at their disposal.
- The Dropbox and especially the Flickr accounts would be updated more frequently with pictures, videos and documents related to the project.

Google Hangouts

The Google Hangouts were continued throughout the Semester. While the Hangouts were held on Tuesdays in the previous semester, this time slot had to change due to conflicts in the time schedules of team members. It was agreed to trial having the Hangouts on Mondays at 09:00 PST (17:00 GMT). Due to further conflicts in schedules the meetings were moved to Wednesdays at 09:00 PST (17:00 GMT). It was discussed by the team members that as Wednesday is mid-week, some important discussions may need to occur on a Monday in order to plan for the week ahead. As such it was agreed to hold the meetings twice weekly. Meetings were held more frequently when specific deadlines approaching.

In order to keep the Hangouts on topic and to cover all of the necessary material during the scheduled time, an agenda was made prior to the commencement of each hangout. During the meeting, the team simply went through the agenda and discussed all of the matters in an efficient manner. Generally an e-mail would be sent to the Global Mailing List prior to a Hangout to confirm the team members’ involvement. While the system is considered ideal by the team due to its versatility and the fact that team members can partake from a variety of locations, the same problems noted in the Fall Semester were again experienced in the Winter Semester. Time lags are often an issue, primarily associated with smartphone use. When an
individual is located in a noisy area, microphones have to be muted to avoid unnecessary noise. Feedback from microphones is also a common issue experienced during Hangouts. Nevertheless, the system provides the team with a platform to communicate remotely, and it is planned to continue to use Google Hangouts frequently in the upcoming Spring Semester.

**Global Mailing List**
When discussing potential communication improvements at the beginning of the Winter Semester, it was agreed to use the Global Mailing List on a more regular basis. One problem noted in the Fall Semester was that some e-mails tended to go ignored, and to overcome this problem, it was agreed by all team members that one designated member from each global team would provide a response to any e-mail in a prompt fashion, no more than 24 hours after receiving it. As such, the e-mail communication maintained by all team members in the Winter Semester was excellent with regular updates occurring throughout the week. The Global Mailing List was especially useful when deadlines approached – when more regular Google Hangouts could not occur due to scheduling conflicts, extensive e-mails were sent between team members. The team will strive to maintain this contact throughout the Spring Semester.

**Team Website** ([www.applauft.com](http://www.applauft.com))
The team website for the project was largely unutilized during the previous Semester. During the first team meeting in the Winter Semester, the team agreed that this should be rectified, and the website should be updated regularly. As such, it was concluded that all findings from any aspect of the project should be communicated through the website. This new process began as early as brainstorming for the Dark Horse prototype, and the website has been frequently updated since. The website was also updated with a more user-friendly blog text editor to facilitate all team members to upload properly formatted and coherent posts. The website was also updated to include a Flickr feed from the team’s Flickr account. The team continued to populate the website with blog entries as the project moved forward into its final stages. (See Next Page)
The team used a Flickr account to share all photos and pictures relevant to the project during the Fall Semester. This method suited all team members due to the ease of access of the account and the ease of sharing photos. The team agreed to continue to use the Flickr account, but to upload photos on a more regular basis. This was especially relevant as prototyping intensified – information can easily get lost in translation if trying to communicate ideas solely through text and the usage of photos helped the team to visualise potential solutions. The Flickr account was used with great frequency during the project.

**Flickr**

Similarly to the way that Flickr was used to share pictures and photos between team members, a shared Dropbox folder was used to share documents and videos that were too large to upload to the Flickr account. Throughout the project, the team used the Dropbox frequently and found very few problems involved with the system.
Communication at Stanford

The Stanford Team had a relatively easy time maintaining daily contact. They have continued to use the GroupMe message forum, which has been active with text, picture, and video messages reliably coming through a few times a day. Email is generally used for longer or more formal messages, and also for summarizing meeting notes, but the majority of work is definitely done in person.

The group started the quarter having two set meeting times per week – Tuesdays and Thursdays at noon– but by the time prototyping began it was clear that meetings of some kind needed to happen basically every day/night of the week. The Loft space has been an amazing resource for the group.

Communication at Trinity

The Trinity group have been in face-to-face contact at least 5 days per week for the duration of the project. When not in face-to-face contact, the team have used a variety of methods to contact each other frequently. Regular e-mail communication was maintained to organise meeting times and to share additional information with one another. The team have continued to organise a forum (found at s13.zetaboards.com/applauft) to keep all information pertaining to the project organised, such as potential benchmarking opportunities, information on needfinding and web links shared with the team. This also prevents nuggets and key teaching team insights from getting lost, since they are instantly written down in an accessible location. Each team member has an idea log on the forum also to keep track of any quick ideas. The team have also set up a Whatsapp group, a smartphone application where all team members can instantly message each other and all team members will see the messages.

Pal Alto SAP Liaisons

Both of the remote teams are in regular contact with their SAP liaisons. The communication is usually carried out via email or on a face-to-face basis. In Palo Alto, the Stanford Team finally procured the SAP Employee ID Badges during the winter quarter, giving them unrestricted entry, which made user testing, interviews, and trips to the SAP coffee machine very easy.
Dublin SAP Liaisons

The team contacted the SAP liaisons frequently. Primarily this occurred to update the liaisons on the progress of the project, but during the Winter Semester, some other key opportunities presented themselves between the team and the SAP liaisons. After hearing the Fall Presentation, the SAP liaisons requested that the Trinity College team visit SAP Dublin to give a presentation to all of the developers in the AppHaus on the 4th of February. This presentation was a modified version of the Fall Presentation that presented all of the work done during the project to date, including the Dark Horse prototype from the Winter Semester. Following this presentation, it presented the team with an ideal opportunity to obtain feedback on all of the work completed to date by potential users. The team emailed the SAP liaisons with a feedback form allowing all developers to give feedback on the prototypes discussed during the presentation as well as to highlight any potential improvements they would like to see incorporated into remote collaboration.

Another primary opportunity presented itself in the form of the visit of Dr Vishal Sikka, CTO of SAP, to the Dublin AppHaus on the 13th of February. As Dr Sikka is a Stanford PhD graduate, he has a keen interest in the design thinking involved in ME310. As such, the SAP liaisons requested that the team give Dr Sikka a presentation of the work completed to date. Again, this presentation was tailored to primarily focus on the prototypes developed thus far in the project. At the conclusion of the presentation, Dr Sikka stated he was very impressed with the progress made by the team to date on the project. He also provided some very useful feedback and benchmarking opportunities to the team.

The SAP liaisons contacted put the group in contact with Valerie Maybin, a business manager in SAP Galway in the west of Ireland, who was very interested in the project after hearing the presentation on the 4th of February. A meeting was set up with Ms Maybin in SAP Dublin to discuss the project on the 7th of March. Ms Maybin was particularly interested in the concept of “turning the AppHaus into the AppStadt”, i.e. remotely connecting every remote AppHaus. She discussed that she would like to see the AppHaus mentality and the products that were discussed during the presentation incorporated into SAP Galway. She shared that design thinking may not be strictly limited to developers, and could have a viable place amongst business employees.
Division of Work

The team endeavoured to divide the work equally between remote locations. While needfinding would inevitably result in different results between the remote locations, the team continued to maintain an equal workload with regard to developing prototypes and submitting assignments, and agreed to share all findings pertaining to the needfinding done in Palo Alto and Dublin.

During the Dark Horse prototype mission, the teams concentrated on different prototypes based on the results of the brainstorming sessions conducted. The Trinity College team concentrated on The SimSAPs prototype and the Stanford University team focused on The Bounty Board. While these prototypes aren’t linked, it gave the team an opportunity to explore a greater number of new ideas and the remote teams actively shared all key findings. As the project advanced towards the FUNKtional prototype, the team began to converge on a central need statement, involving reinventing the Scrum meeting process currently performed in SAP. While the teams again focused on different prototypes (the Scrum-Half and The Scrum Kiosk), the remote teams both concentrated on the Scrum process, and therefore a certain amount of convergence was achieved.

Once the Functional prototype was being developed, the team began to converge even more on the same prototype. Both teams decided to continue to develop the Scrum process, with the Dublin team building a version of a Scrum kiosk based on the needfinding conducted in Dublin. While the prototypes differ slightly, it can be stated that they suit the different cultural needs for each location. This provided the team with a key opportunity to conduct needfinding based on different iterations of the same prototype, and this provided a greater number of learnings. The question of whether the table could be modular and customisable also arose from this specific division of work.

Once the WaterWall concept was converged upon the animation and back end functionality of the system was concentrated on in Dublin, while the booth setup and space design was done in Stanford.
Deliverables & Milestones

**Prelude to Spring Quarter:** The team was very nervous regarding the new direction we had collectively compromised on over spring break in Dublin. We consulted the Teaching Team and the Mentors for advice for our new vision, the Scrum Pole, which was essentially a streamlined, vertical version of the Scrum Table/Scrum Station. We ultimately abandoned the Scrum Pole, though, after 2-3 weeks of toilsome hardware prototyping and lengthy BOM attempts. (Not to mention the fact that “Scrum Pole” might have been the least attractive name ever given to anything. Ever.). During this transition point the team listed out all the discovered Needs and found that the need for casual atmosphere, immediate connection, and serendipitous interactions were all very strong, but we needed a solution that was more in our scope of hardware refinement.

**Spring Hunting Plan:** Over Spring Break the team agreed that our best strategy was to break up into three sub-teams based on technical background and locations. This would help us take more individual responsibility for the myriad challenges ahead of us. The teams were The Peaches (software), The Pears (user-facing hardware), and The Plumbs (technical hardware), and each was divided evenly between Stanford and Trinity Students. The sub-team organization was the central component of our new “hunting plan,” as we had just recently changed the entire direction for our project and were not yet in a place to dive out a specific hardware roadmap, but we did outline a general Gantt Chart and assign certain elements to certain people. The software was the same as our previous prototype, however, so the Peaches, at least, were able to draw up a roadmap of specific milestones. The most effective thing we did happened a week later, when we made our Gannt chart / to-do list publicly visible in our work area, and prominently assign individuals to specific tasks. This helped keep us on track.

**Part X:** Our “Part X” was the WaterWall display interface; this entailed overlaying a live streaming video feed with semi-transparent animations that we had to create ourselves (see next page).
But first, a quick run down on our
New Direction: the Aquarium

The virtual aquarium is a play on the iconic decoration of fish tanks in offices. Two dynamic images are projected onto the wall so they are partially overlapping: a large fish tank, and the camera view from a remote SAP location. It gives the illusion of a real 3D fish tank with a different room on the other side.

Each of the fish is dragging a banner that displays a short word/phrase describing a relevant topic of conversation in the remote location. These are generated daily, via a private social networking platform similar to Twitter. So employees can “tweet” messages to, say, #SAPFISHTANK, and (depending on their popularity) they will be displayed on the banners the following day.

The Aquarium serves several important purposes:

1) To gently attract people to walk by and watch the fish, and catalyzing casual gatherings/conversations amongst local employees.
2) To give viewers bits of information about the remote location being displayed on “the other side,” so that local employees are better-informed about current issues abroad, and so they have topics to start conversations with.
3) To partially obscure the visibility of the other side, so it’s not a privacy invasion.
4) To create an optical illusion of depth (using perspective lines and different widths between the two overlapping images) so that it feels like the other office is really right there.

Manufacturing Plan: By this stage the idea of the Water Wall as a final product had finally been flushed out, at least in terms of hardware. All the hardware components needed for the project were clearly listed and the bill of materials was prepared for EXPE. The team took this opportunity to make a schedule and a to-do list purely for EXPE items; a massive list, which continued to grow from that point on...

The team still did not have an idea on who will be delivering the software elements. Due to the confusion the team decided to go ahead with the programming knowledge it had. A hiring advertisement poster was also made as a fall back, which could later come in handy for hiring computer programmers.
Penultimate Review: The team could not deliver the entire product but delivered all the features/modules separately to the teaching team for review. The team got important feedback from the teaching team on integration of the entire system and the importance of creating our own local area network for EXPE.

The suggestions really helped the team and the team integrated most of the software and hardware components in a week after the Penultimate review. The team learnt an important lesson on time management and making quick design decisions here.
**Budget**

This section summarizes the budget and highlights where the team spent the majority of funds. The comprehensive budget spreadsheet is attached in the Appendix A.

The team spent the money judiciously in the Fall Quarter, spending $550 on prototyping, stationary and team space setup. However, in the Winter quarter the team experimented with a lot of ideas. Most of the money was spent on building Functional and Funky prototypes. The GoPano Panoramic Camera Lens cost around $750 and it was the major component purchased by the team for Scrum Table prototype. The Stanford team invested heavily on the functional and Funky prototype materials, because both our experiences testing at SAP and getting teaching team feedback had continuously pointed us towards refinement and resolution.

Major investment was made in the hardware components of the final EXPE product. The NEC short throw projector system cost $950 and was the major chunk of the budget. Other important hardware components were Modem, wide-angle webcams and Audio system.

Our final budget spreadsheet with expenditures for the year is attached in Appendix 3.
# Stepwise Product User Relationship (SPUR) Theory

Every product has a unique relationship with its user. The depth of that connection and the degree to which it changes over time has a great degree of variation. Whether it is shallow and fleeting or profound and dynamic, every product/user relationship will go through certain stages that have direct analogies in the realm of interpersonal relationships. By considering how each SPUR stage will manifest itself in the context of interaction design, we can reverse-engineer this relationship from the inside out, maximum impact the behavior, emotions, and brand loyalty of our users.

A framework of user scenarios that identifies crucial turning points and distinct stages of intimacy that exist between a product and its user.

<table>
<thead>
<tr>
<th>(#)</th>
<th>SPUR Stage</th>
<th>Distinctions</th>
<th>Emotions</th>
<th>Interpersonal Example</th>
<th>Product/User Example (Phone)</th>
<th>Design Questions</th>
<th>Scenario</th>
<th>Implications for the Design</th>
<th>Relationship Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Life Until</td>
<td>unknown unknowns</td>
<td>unique satisfaction, confusion, incompleteness, naivety</td>
<td>You buy one movie ticket for a romantic comedy.</td>
<td>You carry around your cell phone as a mp3 player and a digital camera at all times.</td>
<td>What specifically is causing classification amongst future users? How do their past experiences affect their current needs and which are explicit vs. implicit? What are their current solutions, and what about those ones do you dislike? Can we satisfy seemingly disparate needs with a &quot;flexible&quot; solution?</td>
<td>Current remote controls: Skype, Cisco, TP (menu review), email, etc. Can we help our corporate clients easily and effectively &quot;plug&quot; our idea to other decision makers at SAP?</td>
<td>Our final presentation, our brochure, our poster, and our inkjet printer will be all our opportunity to plant the &quot;first seed&quot; of SPUR with the corporate clients, and possibly other SAP employees. We have full access to SAP locations, and have the ability to communicate with a handful of employees in four global locations.</td>
<td>Draft actual version of support/personal brochures, which would allow us to provide employees at SAP, and turn the inkjet printer into our &quot;community announcement&quot; board into SAP offices in Paris and Dublin. Test a &quot;wow&quot; prototype at SAP.</td>
</tr>
<tr>
<td>2</td>
<td>The Gossip</td>
<td>known unknowns</td>
<td>curiosity, secret, realization of need, concealed interest, and repressed pursuit.</td>
<td>You stumble upon their profile on your friend's Facebook page.</td>
<td>You watch an iPhone ad while you wait for a YouTube video.</td>
<td>Where/how/when will our user first hear about our product? How might we control the resolution to that question, and use it to our advantage? How might we create an experience for our product before it is introduced at SAP? How might we help our corporate clients easily and effectively &quot;plug&quot; our idea to other decision makers at SAP?</td>
<td>The user first notices our product in the workplace (a): How might we design our product to appear as &quot;lone wolf&quot; during a short coffee break? (b) visible from his/her central workplace. (c) only approachable from the side (e.g. in a hallway). (d) occupied by other employees and activity in area.</td>
<td>(a) Conduct user/astrological research at SAP to analyze flight patterns, and position our products so it captures a popular route. (b) Design the display to be bright, dynamic, vibrant, andArgsSAP, and also visually appealing, while not distracting the user. (c) Include subtle audio cues to engage our users even without visual. (d) Design a gestural interface to be interactive and fun, users should not be standing silly in front of it as they would at a museum gallery.</td>
<td>The Gossip</td>
</tr>
<tr>
<td>3</td>
<td>The Sighting</td>
<td>anonymous, peripheral, indirect</td>
<td>awareness, hope, expectation</td>
<td>You see them in public.</td>
<td>You see someone walking down the street wearing a face mask.</td>
<td>What is the product's first impression of our product as we set it in the context of the workspace? How might we design our product to appear as &quot;lone wolf&quot; during a short coffee break? (a) How might we design our product to appear as &quot;lone wolf&quot; during a short coffee break? (b) visible from his/her central workplace. (c) only approachable from the side (e.g. in a hallway). (d) occupied by other employees and activity in area.</td>
<td>The user is on his/her way to a meeting and happens to walk by our product, he/she stops for a brief moment to take a closer look. There is nobody on &quot;the other side&quot;, so only the face mask is displayed.</td>
<td>The fish respond to the user's presence (e.g. they all jump swimming at once). It quickly becomes clear that neither the fish nor tank are &quot;real&quot;. Upon a closer look around, the user notices a sign on the aquarium saying &quot;Dublin&quot;, and wonders if the other side is actually Dublin. He walks away with mild amusement and curiosity.</td>
<td>The Sighting</td>
</tr>
<tr>
<td>4</td>
<td>The Mutual Glance</td>
<td>brief synchronous awareness, recognition is uncommunicated, conditional</td>
<td>flash judgment, false impression, mutual recognition, intangible</td>
<td>You cross paths at the grocery store and make brief, accidental eye contact.</td>
<td>A friend hands you their iPhone after you agree to take their picture.</td>
<td>How might we continue to build intrigue upon a closer impression? How do the aesthetic changes of a phone from 10AM to 10PM? How might users become apparent that were unclear from afar? How can we surprise and delight the user? What is the &quot;WOW&quot; factor? How might we present an &quot;awed&quot; encounter with the other side?</td>
<td>The user is on his/her way to a meeting and happens to walk by our product, he/she stops for a brief moment to take a closer look. There is nobody on &quot;the other side&quot;, so only the face mask is displayed.</td>
<td>The fish respond to the user's presence (e.g. they all jump swimming at once). It quickly becomes clear that neither the fish nor tank are &quot;real&quot;. Upon a closer look around, the user notices a sign on the aquarium saying &quot;Dublin&quot;, and wonders if the other side is actually Dublin. He walks away with mild amusement and curiosity.</td>
<td>The Mutual Glance</td>
</tr>
<tr>
<td>5</td>
<td>The Handshake</td>
<td>sensory contact, public, prosaic, formal, ambiguous</td>
<td>mutually acknowledged recognition</td>
<td>You get introduced through a mutual friend at an event.</td>
<td>An employee at the Apple Store explains all the new iPhone features are you handle the product on display.</td>
<td>How might we create affinities in the design so that a first-time user knows what to do? What are the gestural &quot;commands&quot; and what happens when they are done by someone else at once? How can we prevent them from happening unintentionally? How might we ensure that users try in proper range of the video camera and microphone?</td>
<td>The user is taking a coffee break and plans on returning to work in roughly 20 minutes. After refilling his/her coffee cup, he casually approaches the tank, which he has previously been in watching from a distance. He is still curious about it. When he notices that a man whom he does not recognize is swimming on the &quot;other side&quot;, he is amazed.</td>
<td>The user notices a &quot;thought bubble&quot; float over the tank and never hear the man in the &quot;other side&quot;. This occurs via our facial recognition feature that links to an employee database. Both sides react to the augmented reality with a gape of surprise, read the info out loud, and the other side confirms its true. The ice is broken.</td>
<td>The Handshake</td>
</tr>
<tr>
<td>6</td>
<td>The Rationalize Date</td>
<td>exclusive interaction, prematurely practiced, proactive, nascently initiated</td>
<td>expressed intimacy with pretense of rationality, desire, excitement, nervousness, judgment</td>
<td>You leave your jacket with them so you can escape to meet up the next day.</td>
<td>You are eligible for an upgrade, and in a 30 day risk free trial, so you decide to buy an iPhone.</td>
<td>What kind of useful functionality can we provide? How might we encourage use of the Think Tank after initial curiosity worn off? How can we ensure the user is not distracted from making the transition of how it works? Today, he is stressed because he needs to make a creative decisions about the project he is working on in order to move forward with his work, but the Telepresence meeting with Dublin is not scheduled for the next Friday.</td>
<td>The user is taking a coffee break and plans on returning to work in roughly 20 minutes. After refilling his/her coffee cup, he casually approaches the tank, which he has previously been in watching from a distance. He is still curious about it. When he notices that a man whom he does not recognize is swimming on the &quot;other side&quot;, he is amazed.</td>
<td>The user notices a &quot;thought bubble&quot; float over the tank and never hear the man in the &quot;other side&quot;. This occurs via our facial recognition feature that links to an employee database. Both sides react to the augmented reality with a gape of surprise, read the info out loud, and the other side confirms its true. The ice is broken.</td>
<td>The Rationalize Date</td>
</tr>
<tr>
<td>7</td>
<td>The Infatuation Period</td>
<td>experiencing the combined aesthetic end of introduction and beginning of relationship excitement, discovery of strengths unknown knowns.</td>
<td>You have pheromonal sex.</td>
<td>You spend all day dating your phone, downloading apps, taking pictures, listening to music... you tell all your friends how great it is.</td>
<td>What is the ideal use of the Think Tank? How often and for what purpose is it used? How long can you and others imagine the user over time after the novelty wears off?</td>
<td>Arnold is quite fond of the Think Tank, as he feels it is almost for several months. He has now experienced all of the interactive display features, and is well-acquainted with many remote colleagues.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>The Letdown</td>
<td>discovery of flaws. Disappointment, disillusionment, resentment, doubt, fear.</td>
<td>He/She forgets your birthday.</td>
<td>You learn that Facebook only works when both parties have text, and the camera view is never flattering.</td>
<td>What are the products shortcomings? How can we prepare for them and not feel let down by long-term trust and support in the user? (a) is the Think Tank on 2.4G, then what happens in the event of malfunctioning hardware / software malfunction? (b) What about device malfunction? (c) How can managers get immediate customer feedback?</td>
<td>Arnold is irritated because James is bugging the Think Tank, using it for all his business meetings. Arnold is annoyed with James because he only uses the Tank for playful, nonproductive purposes.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>The Compromiser</td>
<td>a shared identity is formed. Relationship rules, parameters are established. Public display, lifestyle adapts.</td>
<td>You get married, and agree to bake his/her name. No prenup.</td>
<td>You buy a $50 case to protect your phone from accidental damage, and engrave your name on it.</td>
<td>How might we design rules for the Think Tank so it is equally shared, enjoyed, and available for use by everyone in the office? How can we prevent competition between users and maintain maximum availability, while at the same time encouraging social gatherings? How might we optimize the interaction for 5-20 minutes (no more, no less)?</td>
<td>The Think Tank has its own playful, scientifically controlled timer system. If a person is recognized for longer than 20 minutes, the tank begins to drain, and the fishes slowly die. (depressant on life?) This discourages people from &quot;bogging&quot; the space for too long.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>The Anniversary</td>
<td>ritual and ritualization occur, stereotypes are appropriated, known knowns.</td>
<td>You finish each other’s sentences.</td>
<td>You saw a second phone-shaped logo into your favorite album.</td>
<td>How can we help users create personalized routines of use? What new, adapted behaviors do we hope will develop around the Think Tank? What workarounds do we know? How can we develop a culture around Think Tank users?</td>
<td>The Think Tank is weighed down with un-projects and extraneous add-on features. The sensors are overloaded and dysfunctional. (“Think Tank?” “What do you mean?” “Bluntly your bubble?”/ “I think it’s good.?”)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>The Tipping Point</td>
<td>irreversible damage is done to the relationship</td>
<td></td>
<td>X</td>
<td>You discover he/she cheated, and famously break up.</td>
<td>Drop your iPhone getting out of pocket and make a giant crack on the screen.</td>
<td>How long should the life cycle be for a product like the Think Tank? Is it a question of the hardware wearing out, or of the technology becoming outdated? Is the projected life cycle reflected in the choice of materials / materials / materials / technology / warranty / contract? How might we adopt the product to satisfy the user’s changing needs, thus preventing the breakups as an externality?</td>
<td>The next generation workforce at SAP is considered to be social and collaborative than the current one. New users are too intimidated by experienced users to try it out for themselves.</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>The Rebound</td>
<td>ingoring emotional memories are reflected in future decisions.</td>
<td>You pointedly seek out someone new who is nothing like them.</td>
<td>You consider switching to Android, but your music, calendar, photos, etc. are all in iOS software. Plus, you’re eligible for another upgrade.</td>
<td>After the Think Tank is no longer in use, how will the buyer replace it? Will they buy the newest Cisco product or remain loyal to the AppleTalk brand? Can they upgrade to the next generation Think Tank 2.0? How might we incentivize the latter?</td>
<td>The Think Tank hardware warranty contract is for 2 years, which is when your software will wear out, but it’s roughly the time when we predict the buyer will desire an upgrade. So after the 2 year contract is up, the buyer becomes eligible for a discounted “upgrade” to the next generation Think Tank 2.0, for which development is already underway. (But “The Gospel” begins...)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EMBRACING SERENDIPITY

APPLEAUFT BLOWS CISCO TELEPRESENCE OUT OF THE WATER WITH A "BLUE OCEAN" STRATEGY

SAP takes a deep dive into team saseology & finds "Interactions are like fish. They have many scales."

“WE DON'T CARE WHAT THEY TALK ABOUT, AS LONG AS THEY ARE TALKING” - THE TAO

Finally, a video call that isn’t awkward!” - Mark Zuckerberg

Trinity College & Stanford University TEAM UP TO DESIGN

“Somebody get me some stock options!” - Mark Schar
Creative turning points are often born out of serendipitous interactions. An hour spent at the local water cooler is much more likely to spark new inspiration than an hour at a solitary desk. Many booming tech companies have realized this, and started building “campuses” instead of offices. They offer boundless on-site perks to keep people at work as long as possible, because unfortunately, this phenomenon only works within the “campus” walls. It seems that even in a digital world, there is no substitute for face-to-face interaction.

Remote collaboration in all industries remains an enormous challenge. Expensive videoconferencing tools are well-suited for presentations and structured meetings, but their formality stifles creativity and they take weeks to schedule. Webcam video-chats have a more relaxed atmosphere, but the narrow view, tiresome setup, and awkward interactions make it an equally unappealing option overall. These issues are exacerbated when the people on either end are unfamiliar with each other in the first place.

These are the problems that inspired us to create The WaterWall. It has a live 24/7 video-stream to ensure that no time is wasted on setup, an interactive graphic display to make interactions more playful and enjoyable, and a "thought bubble" generator that displays current projects that are relevant to both parties. These subtle features enable The WaterWall to spark new intersections and unforeseen collaborations between different teams, departments, and locations across SAP’s vast global network.
**EMBRACING SERENDIPITY**

The realistic 3D aquarium serves as an aesthetic ‘bridge’ to soften the transition between the two remote environments. The system provides audio and speech capability, in essence acting as a life-size videochat system. The dynamic aesthetic encourages passersby to approach the WaterWall spontaneously, promoting casual encounters between colleagues who may never cross paths otherwise.

**BREAKING the ICE**

The WaterWall is a place where people will want to be; the aquarium is pleasant to look at, and the photo-realistic fish react to the sounds, presence and motion of those in the vicinity. “Thought bubbles” come subtly from the fish – these provide a snapshot of the corporate SAP mind, highlighting current projects and big topics at that location. Viewers can interact with the “thought bubbles” and reveal more information by simply pointing at them. Well-acquainted users have the option of lowering the aquarium’s opacity for a more direct face-to-face chat.
#! /usr/bin/env python2

# Primary Author: Mark Culleton
# Email: culletom@tcd.ie
# Co-Author: Raymond Barrett
# Email: barretrj@tcd.ie

import time
import os
import math
import random
import pygame
import pygame.camera
from pygame.locals import *
import pyganim
from wordle import WordleWord
from frame_get import get_frame, string2Surface
import cv2.cv as cv
import cv2
import Image
import numpy as np
from Gestures import GestureTracker

NUM_FISH = 5
DEPTH = 30
SCREENSIZE = [1024,768]
GRAVITY = 9.81  #m/s^2
BUYOANCYT = 9.2    #m/s^2
WATER_RESIS = 0.2 #L/s

WORDS = ['Images/Words/word%s.png'%(n) for n in xrange(NUM_FISH)]
# COLLISIONS = [0 for n in xrange(NUM_FISH)]
# print COLLISIONS
FISH_POSITIONS = []
FISH_DIRECTIONS = []

DRAW = cv.CreateImage(SCREENSIZE, cv.IPL_DEPTH_8U, 3)
smallres = (SCREENSIZE[0]/5, SCREENSIZE[1]/5)
smallscreenpos = (SCREENSIZE[0]-smallres[0]-10,SCREENSIZE[1]-smallres[1]-10)
cascade = cv.Load("haarcascade_frontalface_default.xml")
image_res2 = (512, 400)
COUNT = 2

DETECTSIZE = (SCREENSIZE[0]/6,SCREENSIZE[1]/6)
POINTERSIZE = [24,34]

--------------------
def surface_to_string(surface):
    """Convert a pygame surface into string"""
    return pygame.image.tostring(surface, 'RGB')

def pygame_to_cvimage(surface):
    """Convert a pygame surface into a cv image"""
    image_string = surface_to_string(surface)
    cv_image = cv.CreateImage(surface.get_size(), cv.IPL_DEPTH_8U, 3)
    cv.SetData(cv_image, image_string)
    return cv_image

def string_to_cvimage(string):
    """Convert a pygame surface into a cv image"""

-------------------
cv_image = cv.CreateImage(image_res2, cv.IPL_DEPTH_8U, 3)
cv.SetData(cv_image, string)
image_bgr = cv.CreateMat(cv_image.height, cv_image.width, cv.CV_8UC3)
cv.CvtColor(cv_image, image_bgr, cv.CV_RGB2BGR)
big_image = cv.CreateImage(SCREENSEIZE, cv.IPL_DEPTH_8U, 3)
cv.Resize(image_bgr, big_image)
return big_image

def string_to_cv2image(string):
    """Convert image string to CV2 image""
    #print string
    img = np.fromstring(string, dtype=np.uint8)
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    #print img
    #print type(img)
    return img

def cvimage_grayscale(cv_image):
    """Converts a cvimage into grayscale""
    grayscale = cv.CreateImage(cv.GetSize(cv_image), 8, 1)
cv.CvtColor(cv_image, grayscale, cv.CV_RGB2GRAY)
return grayscale

def cvimage_to_pygame(image):
    """Convert cvimage into a pygame image""
    image_rgb = cv.CreateMat(image.height, image.width, cv.CV_8UC3)
cv.CvtColor(image, image_rgb, cv.CV_BGR2RGB)
return pygame.image.frombuffer(image.tostring(), cv.GetSize(image_rgb),"RGB").convert()
points = detect_faces(cv_image, storage)  # Get points of faces.

return (points, point, gesture)

class Fish(pygame.sprite.Sprite):
    def __init__(self, num, word):
        pygame.sprite.Sprite.__init__(self)
        self.num = num
        self.angle = 0.0
        self.count = 0  # Used to control duration of frames in sprite animation
        self.d_x = 0; self.d_y = 0; self.d_z = 0
        self.face_counter = 0  # Used to reset facial detection and account for occasional loss in detection
        self.scale_counter = 0
        self.sideward_images = []
        self.turn_images = []
        self.forward_images = []
        self.half_turn_backward = []  # Going from forward swimming to side
        self.clock = pygame.time.Clock()
        self.path = 'Images/Fish_GIF/Fish%s/'%self.num
        self.word = WordleWord(word)
        self.detected_angle = 0
        self.Detected_Face = False
        self.Reacted_To_Face = False
        self.Turn = False
        self.Swim_Forward = False
        self.Transparent = False
        self.Rotated_Forward = False
        self.Rotated_Sideward = False
        self.Avoiding_Fish = False
        self.after_initial_loop = False
        self.motion = 'Turn'
        path, dirs, self.turn_frames = os.walk(self.path + self.motion).next()  # Create each fish sprite ('filename of each frame, duration_of_frame_in_seconds)
        for i in xrange(1, len(self.turn_frames)):  
            self.turn_images.append((self.path + self.motion + '/frame_%s.png'%(i)))
            halfway = int(len(self.turn_images)/2)
            for i in xrange(halfway,len(self.turn_images)-1):
                self.half_turn_backward.append(self.turn_images[i])
        self.motion = 'Forwards'
        path, dirs, self.forward_frames = os.walk(self.path + self.motion).next()  # Create each fish sprite ('filename of each frame, duration_of_frame_in_seconds)
        for i in xrange(1, len(self.forward_frames)):  
            self.forward_images.append((self.path + self.motion + '/frame_%s.png'%(i)))
        self.motion = 'Sidewards'
        path, dirs, self.forward_frames = os.walk(self.path + self.motion).next()  # Create each fish sprite ('filename of each frame, duration_of_frame_in_seconds)
        for i in xrange(1, len(self.forward_frames)):  
            self.sideward_images.append((self.path + self.motion + '/frame_%s.png'%(i)))

        self.fsh = pyganim.PygAnimation(self.sideward_images, 0.05)
        self.width, self.height = self.fsh.getMaxSize()
        self.x_pos = random.uniform(self.width, SCREENSIZE[0]-self.width)
        self.y_pos = random.uniform(self.height, SCREENSIZE[1]-self.height)
        self.z_pos = random.randint(1, DEPTH)
        # X-Direction(+1:Left-to-Right, -1: Right-to-Left)  Y-Direction (+1: Down, -1: Up)  Z-Direction(+1: Backwards, -1:Forwards)
        self.x_dir = random.choice([1,-1])
        self.y_dir = random.choice([1,-1])
        self.z_dir = random.choice([1,-1])
self.impulse_vel = random.uniform(60, 100)  # Initial Velocity of fish caused by fin motion
self.angle = random.uniform(0, 30) * math.pi / 180  # Angle fish swims (In Radians)
self.x_speed = self.x_dir*(self.impulse_vel * math.cos(self.angle))  # Magnitude of Velocity in x-direction
self.y_speed = self.y_dir*(self.impulse_vel * math.sin(self.angle))  # Magnitude of Velocity in y-direction
self.z_speed = self.z_dir*random.uniform(2,5)
self.x_acc = -(WATER_RESIS * self.x_speed)
self.y_acc = -(WATER_RESIS * self.y_speed) - (GRAVITY-BUYOANCY)
self.z_acc = -(WATER_RESIS * self.z_speed)

##NOTE: '-' implies fish moving up and '+' implies fish moving down
if self.y_speed > 0:  # Fish moving Downwards (y_speed is +ive)
    self.y_acc = -(WATER_RESIS * self.y_speed) - (GRAVITY-BUYOANCY)
else:  # Fish moving Upwards (y_speed is -ive)
    self.y_acc = -(WATER_RESIS * self.y_speed) + (GRAVITY-BUYOANCY)
self.crit_vel = self.impulse_vel / 2  # Velocity at which fish flaps again
FISH_POSITIONS.append((self.x_pos, self.y_pos))  # Globally store each fish location
FISH_DIRECTIONS.append((self.x_dir, self.y_dir))  # Globally store each fish location

if self.x_dir < 0:
    self.fsh.flip(True, False)
    self.fsh.rotate(-self.angle)
self.fsh.play()  # Begins playing the fish sprite

def update(self):
    self.time_passed = self.clock.tick()  # Unit is milliseconds
    self.time = self.time_passed / 1000.0  # Unit is seconds
    if (self.x_pos + self.width > SCREENSIZE[0] and self.x_dir > 0) or (self.x_pos < 0 and self.x_dir < 0):
        if self.Swim_Forward:
            self.x_speed *= -1  
            self.x_dir *= -1
        else:
            self.Turn = True
    if (self.y_pos + self.height > SCREENSIZE[1]-40 and self.y_dir > 0) or (self.y_pos < 0 and self.y_dir < 0):
        self.y_dir *= -1
        self.y_speed *= -1
        self.angle *= -1
        if self.z_pos >= DEPTH or self.z_pos <= 0:
            self.z_speed *= -1
            self.z_dir *= -1
            if self.num < 5:
                # if math.fabs(self.z_pos) >= DEPTH:
                self.Swim_Forward = True
                self.Swim_Backward = True
            else:
                self.Swim_Forward = True
                self.Swim_Backward = True
        self.check_for_collision()
    self.d_x = (self.x_speed * self.time) + (0.5 * self.x_acc * math.pow(self.time, 2))  # Current displacement is calculated using previously recorded values
    self.d_y = (self.y_speed * self.time) + (0.5 * self.y_acc * math.pow(self.time, 2))
    self.d_z = (self.z_speed * self.time)
    if self Detected_Face and not self.Reacted_To_Face:
        self.detected_face()
    elif self.Turn:
        self.turn()
    elif self.Swim_Forward:
        self.swim_forward()
    else:
        self.check_for_collision()
        self.normal_swim()
    # self.rotate()
    # if self.scale_counter % 5 == 0:
    #     self.scale()
    self.scale_counter += 1
self.x_acc = -(WATER_RESIS * self.x_speed)
self.z_acc = -(WATER_RESIS * self.z_speed)
if self.y_speed > 0: #Fish moving Downwards (y_speed is +ive)
    self.y_acc = -(WATER_RESIS * self.y_speed) - (GRAVITY-BUYOANCY)
else: #Fish moving Upwards (y_speed is -ive)
    self.y_acc = -(WATER_RESIS * self.y_speed) + (GRAVITY-BUYOANCY)
self.x_speed += self.time*self.x_acc # v=u+at
self.y_speed += self.time*self.y_acc # Since up is negative and down is positive
self.z_speed += self.time*self.z_acc
self.x_pos += self.d_x
self.y_pos += self.d_y
self.z_pos += self.d_z

FISH_POSITIONS[self.num] = (self.x_pos,self.y_pos)
FISH_DIRECTIONS[self.num] = (self.x_dir,self.y_dir)

def impulse(self):
    self.angle = random.uniform(0,30) * math.pi / 180 # Angle fish swims (In Radians)
    self.impulse_vel = random.uniform(60,100)
    self.y_dir = random.choice([-1,1])
    self.z_dir = random.choice([-1,1])
    self.x_speed = self.x_dir*(self.impulse_vel * math.cos(self.angle)) # Magnitude of Velocity in x-direction
    self.y_speed = self.y_dir*(self.impulse_vel * math.sin(self.angle)) # Magnitude of Velocity in y-direction
    self.z_speed = self.z_dir*random.uniform(2,5)
    self.count = 0 # Reset the counter
    self.fsh.rate = 1.2 # Speed up animation frames to coincide with increased speed

def normal_swim(self):
    if math.fabs(self.x_speed) < self.crit_vel: #Fish is now swimming too slow
        self.impulse()
    else:
        if self.count > 10: # Once count has exceeds 10, animation returns to normal speed
            self.fsh.rate = 0.4
            self.count += 1

def turn(self):
    if self.motion != 'Turn': #Reset
        self.motion = 'Turn'
        self.fsh.change_animation(self.turn_images, 0.02, loop = False)
        self.fsh.rate = 1.2
        self.z_speed = 0
        self.Changed_Dir = False
        if self.x_dir < 0:
            self.fsh.flip(True, False)
            current_frame = self.fsh.current_frame_num
            if current_frame < len(self.turn_images) / 2 and not self.Changed_Dir:
                self.x_speed *= 0.5
                self.y_speed *= 0.5
            else:
                if not self.Changed_Dir:
                    if self.Detected_Face:
                        self.angle = self.detected_angle
                        self.x_speed = math.copysign(self.impulse_vel/2 * math.cos(self.angle),-self.x_speed)
                        self.y_speed = math.copysign(self.impulse_vel/2 * math.cos(self.angle),self.y_speed)
                        self.Changed_Dir = True
                        if self.fsh.state == 'stopped':
                            self.Turn = False
                            self.x_dir = -1
                            self.motion = 'Sidewards'
                            self.fsh.change_animation(self.sideward_images, 0.05, loop = True)
                            if self.x_dir < 0:
                                self.fsh.flip(True, False)
                                self.impulse()
def swim_forward(self):
    if self.motion != 'Turn' and not self.Rotated_Forward:
        self.motion = 'Turn'
        self.fsh.change_animation(self.turn_images, 0.02, loop = False)
        self.fsh.rate = 1.2
        self.Rotated_Sideward = False
        if self.x_dir < 0:
            self.fsh.flip(True, False)
        current_frame = self.fsh.current_frame_num
        if current_frame < len(self.turn_images) / 2 and not self.Rotated_Forward:
            self.x_speed *= 0.5
            self.y_speed *= 0.5
        else:
            if not self.Rotated_Forward:
                self.motion = 'Forwards'
                self.fsh.change_animation(self.forward_images, 0.05, loop = True)
                self.fsh.rate = 0.8
                self.Rotated_Forward = True
            elif self.z_pos <= 5:
                if not self.Rotated_Sideward:
                    self.motion = 'Half_Turn'
                    self.fsh.change_animation(self.half_turn_backward, 0.05, loop = False)
                    self.fsh.rate = 1.2
                    self.Rotated_Sideward = True
                    if self.x_dir > 0:
                        self.fsh.flip(True, False)
                    if self.fsh.state == 'stopped':
                        self.Rotated_Forward = False
                        self.Rotated_Sideward = False
                        self.Swim_Forward = False
                        self.motion = 'Sidewards'
                        self.fsh.change_animation(self.sideward_images, 0.05, loop = True)
                        if self.x_dir < 0:
                            self.fsh.flip(True, False)
                        self.impulse()
                else:
                    if math.fabs(self.z_speed) < 2: #Fish is now swimming too slow
                        self.z_speed = -random.uniform(5,10)

    def swim_to_point(self, points):
        self.face_rects = []
        if not self.Detected_Face:
            self.Detected_Face = True
            for i in range(len(points)):
                p, _ = points[i] #At present only reacts to one face
                x,y,w,h = p
                x,y,w,h = scale_points(x,y,w,h)
                self.face_rects.append([x,y,w,h])
                if self.y_pos+self.height/2 < y+h/2:
                    multiplier = -20*self.num
                else:
                    multiplier = 20*self.num
                self.face_rect = ((x,y),(x+w,y+h+multiplier))
            self.face_center = (x+w/2, y+h/2)

    def detected_face(self):
        print 'Face Center:', self.face_center
    if self.Swim_Forward:
        print "IN SWIM FORWARD"
        self.Reacted_To_Face = True

    else:
        print 'Face Center:', self.face_center
    if self.Swim_Forward:
        print "IN SWIM FORWARD"
        self.Reacted_To_Face = True
self.swim_forward()

elif (self.x_pos + self.width/2 > self.face_rect[0][0]) and (self.x_pos + self.width/2 < self.face_rect[1][0]) and (self.y_pos + self.height/2 > self.face_rect[0][1]) and (self.y_pos + self.height/2 < self.face_rect[1][1]):
    print "IN TARGET AREA"
    self.Swim_Forward = True
    self.Reacted_To_Face = True
    self.impulse_vel = random.uniform(60,100)
    self.x_speed *= 0.25
    self.y_speed *= 0.25
    self.z_pos = DEPTH-2
    self.z_dir = -1
    self.z_speed = -5
    self.swim_forward()

elif self.Turn:
    self.turn()

elif not self.after_initial_loop:
    self.after_initial_loop = True
    self.detected_angle = -math.atan2((self.face_center[1] - self.y_pos+self.height/2),(self.face_center[0] - self.x_pos+self.width/2))    #Angle between two points
    print self.detected_angle * 180/math.pi
    if math.fabs(self.detected_angle * 180/math.pi) > 80 and math.fabs(self.detected_angle * 180/math.pi) <= 90:
        self.detected_angle -= math.copysign(10*math.pi/180,self.detected_angle)
    elif math.fabs(self.detected_angle * 180/math.pi) > 90 and math.fabs(self.detected_angle * 180/math.pi) < 100:
        self.detected_angle += math.copysign(10*math.pi/180,self.detected_angle)
        print "IN BETWEEN 90 AND 100"
        print self.detected_angle * 180/math.pi
    if math.fabs(self.detected_angle * 180/math.pi) > 90:
        print "IN GREATER THAN 90"
        if self.x_dir > 0:
            self.Turn = True
        if self.detected_angle < 0:
            self.detected_angle = (-180*math.pi/180) - self.detected_angle
        else:
            self.detected_angle = (180*math.pi/180) - self.detected_angle
        else:
            print "IN ELSE STATEMENT"
            if self.x_dir < 0:
                self.Turn = True

    print self.detected_angle * 180/math.pi

    if self.Turn:
        print "SENT TO TURN"
        self.impulse_vel = random.uniform(60,100)
        self.x_speed *= 0.5 # Magnitude of Velocity in x-direction
        self.y_speed *= 0.5 # Magnitude of Velocity in x-direction
        self.turn()

    else:
        if math.fabs(self.x_speed) < 2 * self.crit_vel: #Fish is now swimming too slow
            self.impulse_vel = random.uniform(100,120) * 2
            self.y_dir = math.copysign(1,-self.detected_angle)
            self.x_speed = self.x_dir*(self.impulse_vel * math.cos(self.detected_angle)) # Magnitude of Velocity in x-direction
            self.y_speed = self.y_dir*(self.impulse_vel * math.sin(math.fabs(self.detected_angle))) # Magnitude of Velocity in y-direction
            self.z_speed = 5
            self.fsh.rate = 1.5 # Speed up animation frames to coincide with increased speed
def reset_detection(self):
    self.Detected_Face = False
    self.Reacted_To_Face = False
    self.after_initial_loop = False

def check_for_collision(self):
    for i in range(len(FISH_POSITIONS)):
        x_diff = math.fabs(self.x_pos - FISH_POSITIONS[i][0])
        y_diff = math.fabs(self.y_pos - FISH_POSITIONS[i][1])
        if (x_diff < self.width * 1.2) and (y_diff < self.height):
            if self.x_dir == FISH_DIRECTIONS[i][0]:
                if (self.y_pos < FISH_POSITIONS[i][1]):  # Fish is above other fish
                    self.y_dir = -FISH_DIRECTIONS[i][1]
                    self.y_speed = math.copysign(self.y_speed, self.y_dir)
            elif (y_diff < self.height/2) and (x_diff > self.width/2):
                if self.x_pos > FISH_POSITIONS[i][0]:
                    if (self.x_dir == -1 and FISH_DIRECTIONS[i][0] == 1):
                        self.Turn = True
                if self.x_pos < FISH_POSITIONS[i][0]:
                    if (self.x_dir == 1 and FISH_DIRECTIONS[i][0] == -1):
                        self.Turn = True
                else:
                    self.y_dir = -FISH_DIRECTIONS[i][1]
                    self.y_speed = math.copysign(self.y_speed, self.y_dir)
            FISH_DIRECTIONS[self.num] = (self.x_dir, self.y_dir)
        else:
            self.y_dir = -FISH_DIRECTIONS[i][1]
            self.y_speed = math.copysign(self.y_speed, self.y_dir)

    def check_for_interaction(self, point):
        if (math.fabs(self.x_pos + self.width/2 - point[0]) < 50) and (math.fabs(self.y_pos + self.height/2 - point[1]) < 50):
            return True
        else:
            return False

def scale(self):
    self.width, self.height = self.fsh.getMaxSize()
    ratio = self.width/self.height
    self.fsh.scale((self.width - int(self.z_pos), self.height - int(self.z_pos/ratio)), self.motion, self.x_dir)
    if self.transparent:
        self.fsh.make_transparent(0)
    if self.x_dir < 0:
        self.fsh.flip(True, False)

def rotate(self):
    if math.fabs(self.angle - self.desired_angle) > 0.1:
        if self.desired_angle > self.angle:
            self.fsh.rotate(-(self.angle + 0.05))
            self.angle += 0.05
        else:
            self.fsh.rotate(-(self.angle - 0.05))
            self.angle -= 0.05
    self.Transparent:
    self.fsh.make_transparent(0)
    if self.x_dir < 0:
        self.fsh.flip(True, False)

def blit(self, screen):  # Pastes fish onto screen
    self.screen = screen
if not self.Transparent:
    self.fsh.blit(screen, self.get_pos())

def get_pos(self):
    return (self.x_pos, self.y_pos)

def make_transparent(self):
    self.fsh._visibility = False  # Calls the pyganim.py method
    self.Transparent = True

def reset_colour(self):
    self.fsh._visibility = True
    self.Transparent = False

class Bubble(pygame.sprite.Sprite):
    def __init__(self, word):
        pygame.sprite.Sprite.__init__(self)
        self.clock = pygame.time.Clock()
        self.img = []
        for i in xrange(1, 26):
            self.img.append('Images/Bubble_GIF/Bubble2_%s.png' % (i))  # Load Each Frame
        self.bub = pyganim.PygAnimation(self.img, 0.2)
        self.width, self.height = self.bub.getMaxSize()
        self.bub.play()  # Play the Bubble Animation

    def blit(self, screen, pos):  # Paste Bubble animation onto screen
        self.bub.blit(screen, pos)

    def make_transparent(self):
        self.bub._visibility = False

    def reset_bubble(self):
        self.bub._visibility = True

class VideoFishTank(object):
    def __init__(self, **argd):
        self.__dict__.update(**argd)
        super(VideoFishTank, self).__init__(**argd)
        self.screen = pygame.display.set_mode(SCREENSIZE)
        pygame.display.set_caption('The SAP Fish Tank')
        pygame.display.toggle_fullscreen()
        self.orig_air_bubble = pygame.image.load('Images/Bubble1.png').convert_alpha()
        p = pygame.image.load('Images/mouse-cursor-icon.png').convert_alpha()
        self.pointer = pygame.transform.scale(p, POINTERSIZE)

        self.click_count = 0
        self.transparent_bub = False
        self.previousscreen = None
        self.background = None
        self.emit_word = []
        self.transparency = []
        self.air_bubble = []
        self.bubble_location = []
        self.bubble_size = []
        self.word = []
        for i in range(NUM_FISH):
            self.emit_word.append(False)
            self.transparency.append(255)
            self.air_bubble.append(self.orig_air_bubble)
            self.bubble_size.append(5)
            self.bubble_location.append([0, 0])
            self.word.append(pygame.image.load(WORDS[i]).convert_alpha())

        self.clock = pygame.time.Clock()
self.count = COUNT

def setup_ocean(self):  # These remain stationary, therefore can be recalled when resetting fish tank transparency
    ocean_path = 'Images/Ocean_GIF/Ocean1/'
    ocean_images = []
    path, dirs, ocean_frames = os.walk(ocean_path).next()
    for i in xrange(1, len(ocean_frames)):  # Create each fish sprite ('filename of each frame', duration_of_frame_in_seconds)
        ocean_images.append(ocean_path + 'frame_%s.png' % (i))
    self.ocean = pyganim.PygAnimation(ocean_images, 0.05)
    self.ocean.scale(SCREENSIZE, 'Ocean', 1)
    self.ocean.make_transparent(100)
    self.ocean.play()

def setup_foreground(self):
    self.foreground = pygame.image.load('Images/Fish_Tank_Bed2.png').convert_alpha()
    self.foreground = pygame.transform.scale(self.foreground, SCREENSIZE)
    self.foreground.fill((255, 255, 255, 255), None, pygame.BLEND_RGBA_MULT)

def setup_fish(self):  # Can only be used for initial setup, as fish are moving
    self.fish = [Fish(n, 'word%d' % n) for n in xrange(NUM_FISH)]    # Load the chosen number of fish

def setup_bubbles(self):
    self.bubble = Bubble(None)

def make_transparent(self):  # Makes full fish tank overlay transparent
    self.foreground.fill((255, 255, 255, 0), None, pygame.BLEND_RGBA_MULT)
    self.ocean._visibility = False
    self.bubble.make_transparent() 
    self.transparent_bub = False
    for fsh in self.fish:
        fsh.make_transparent()

def reset_tank(self):  # Remove transparency and return fish tank overlay
    self.ocean._visibility = True
    self.setup_foreground()
    self.bubble.reset_bubble()
    self.transparent_bub = False
    for fsh in self.fish:
        fsh.reset_colour()

def update(self, queue):  # Iterative Loop
    point = []
    points = []
    smallscreen = None
    #for i in range(50):
    #    self.depth[i].fill(255)
    #    self.depth[i].set_alpha(255)
    data = get_frame(self.address)

    # if self.count > 0:
    #    background = string2Surface(data)
    #    self.count -= 1
    # else:
    #    background = faceRecogStuff(data)
    #    self.count = COUNT
    # self.background = string2Surface(data)

    # Pull local frame off queue
    if queue.full():
        smallscreen = queue.get()
        points, point, gesture = FaceAndGestureRecognition(smallscreen)
        smallscreen = string2Surface(smallscreen)
if gesture == 0 and self.state == 'tank':
    # print gesture, self.state
    self.check_event(point)
elif gesture == 5 and self.state == 'word':
    # close wordle
    #print gesture, self.state
    self.check_event((0,0))
    print "GESTURE 5"
elif gesture == 5 and self.state == 'user':
    # close wordle
    #print gesture, self.state
    self.check_event((0,0))
    print "GESTURE 5"
elif gesture == 3 and self.state == 'word':
    # print gesture, self.state
    #print "GESTURE 3"
    self.check_event(point)
    print gesture, self.state

    # Make it the smaller size
    smallscreen = pygame.transform.scale(smallscreen, smallres)
else:
    smallscreen = self.previouscreen

    # Paste all layers on screen
    self.screen.blit(self.background,(0,0))
    self.ocean.blit(self.screen,(0,0))  # Used if Ocean is animated
    self.screen.blit(self.foreground,(0,0))
    self.bubble.blit(self.screen, (100,0))
    if not self.transparent_bub:
        if random.randint(0,20) == 5:
            self.choice = random.randint(0, NUM_FISH-1)
            self.chosen_fish = self.fish[self.choice]
            if self.chosen_fish.y_pos < SCREENSIZE[1]/3 or self.chosen_fish.x_pos > SCREENSIZE[0]-150 or
                self.chosen_fish.x_pos < 150:
                pass
            elif self.emit_word[self.choice] == True:
                pass
            else:
                self.transparency[self.choice] = 255
                self.emit_word[self.choice] = True
                self.bubble_size[self.choice] = 5
                if (self.chosen_fish.x_speed < 0):
                    self.bubble_location[self.choice] = [self.chosen_fish.x_pos, self.chosen_fish.y_pos + self.chosen_fish.height/2]
                else:
                    self.bubble_location[self.choice] = [self.chosen_fish.x_pos + self.chosen_fish.width, self.chosen_fish.y_pos +
                        self.chosen_fish.height/2]

            for i in xrange(len(self.emit_word)):
                if self.emit_word[i]:
                    self.emit(i)
            else:
                for i in range(NUM_FISH):
                    self.emit_word[i] = False

        for fish in self.fish:
            fish.blit(self.screen)
            if points:
                fish.swim_to_point(points)
                fish.face_counter = 0
            elif fish.face_counter > 30:
                pass
            else:
                fish.face_counter += 1
    else:
        pass

    # If user touches fish:
    for i in xrange(len(self.emit_word)):
        if self.emit_word[i]:
            self.emit(i)
    else:
        for i in range(NUM_FISH):
            self.emit_word[i] = False

for fish in self.fish:
    fish.blit(self.screen)
fish.reset_detection()
else:
    fish.face_counter += 1
    fish.update()

if smallscreen:
    self.previousscreen = smallscreen
    self.screen.blit(smallscreen, smallscreenpos)
else:
    self.screen.blit(self.previousscreen, smallscreenpos)

if self.chk >= 0 and self.state == 'word':
    self.fish[self.chk].word.display(self.screen)
elif self.chk >= 0 and self.state == 'user':
    print self.av
    self.fish[self.chk].word.associated[self.av].display(self.screen)

if type(point) == tuple and gesture >= -1:
    self.screen.blit(self.pointer, point)

self.clock.tick()
# print self.clock.get_fps()

pygame.display.update()

def emit(self, choice):
    self.air_bubble[choice] = self.orig_air_bubble
    self.air_bubble[choice] = pygame.transform.scale(self.air_bubble[choice], (self.bubble_size[choice], self.bubble_size[choice]))
    self.air_bubble[choice].fill((255, 255, 255, self.transparency[choice]), None, pygame.BLEND_RGBA_MULT)
    self.bubble_location[choice][1] -= 8
    self.screen.blit(self.air_bubble[choice], self.bubble_location[choice])
    self.bubble_size[choice] += 2
    if self.bubble_size[choice] <= 10:
        self.bubble_location[choice][0] += self.fish[choice].d_x
    else:
        self.word[choice] = pygame.image.load(WORDS[choice]).convert_alpha()
        self.word[choice].fill((255, 255, 255, self.transparency[choice]), None, pygame.BLEND_RGBA_MULT)
        self.word_width, self.word_height = self.word[choice].get_size()
        self.bubble_width, self.bubble_height = self.air_bubble[choice].get_size()
        self.scaler = float(self.word_width) / float(self.bubble_size[choice])
        self.word[choice] = pygame.transform.scale(self.word[choice], (int(self.word_width / self.scaler), int(self.word_height / self.scaler)))
        self.screen.blit(self.word[choice], (self.bubble_location[choice][0], self.bubble_location[choice][1] + (self.bubble_height / 2 - int(self.word_height / self.scaler)) / 2))
        if self.bubble_size[choice] >= 60:
            self.bubble_size[choice] -= 2
            self.transparency[choice] -= 10
            if self.transparency[choice] < 0:
                self.emit_word[choice] = False

def check_event(self, event):
    i = 0
    self.av = -1
    if self.state == "tank":
        for i in range(len(self.fish)):
            chk = self.fish[i].check_for_interaction(event)  # Check if click is over a fish
            if chk:
                self.state, self.av = self.fish[i].word.change_state(self.state, event)
                self.chk = i
else:
self.state, self.av = self.fish[self.chk].word.change_state(self.state, event)

def main(self, addr, que):
    self.setup_ocean()
    self.setup_foreground()
    self.setup_fish()
    self.setup_bubbles()

    self.address = addr
    self.av = -1
    self.chk = -1
    self.state = 'tank'

    while True:  #Game Loop - Detects Key Presses and Mouse Clicks
        for event in pygame.event.get():
            if event.type == QUIT or pygame.key.get_pressed()[K_ESCAPE]:
                pygame.display.quit()
                exit()
            elif pygame.key.get_pressed()[K_UP]:
                self.make_transparent()
            elif pygame.key.get_pressed()[K_DOWN]:
                self.reset_tank()
            elif event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:  #Left Clicks
                    self.check_event(event.pos)
                    self.update(que)

def main():
    pygame.init()
    VideoFishTank().main()
    pygame.quit()

    if __name__ == "__main__":
        main()
#!/usr/bin/env python2

# Author: Raymond Barrett
# Email: barretrj@tcd.ie

import socket
import pygame
import Image
import sys
from time import sleep

#image_res = (1040,680)
image_res = [1024,768]
#image_res = (1280,720)
#image_res = (1900,1080)
image_res2 = (512, 384)
#chunk_size = 8192
#chunk_size = 4096
chunk_size = 2048

# Functions ================================================================
def string2Surface(data):
    # We turn the data we received into a PIL image
    image = Image.fromstring("RGB", image_res2, data)
    # Turn the PIL image into a pygame surface
    image = pygame.image.frombuffer(image.tostring(), image_res2, "RGB").convert()
    image = pygame.transform.scale(image, image_res)
    return image
#==============================================================================
def get_frame(address):
    # not_connected = True
    # Create a socket connection for connecting to the server
    while not_connected:
        try:
            # Create a socket connection for connecting to the server
            client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            client_socket.connect(address)
            not_connected = False
        except:
            # print "Waiting for connection"
            sleep(1)
            buff = client_socket.recv(chunk_size)
            # print buff
            image_size, _, data = buff.partition('::')
            image_size = int(image_size)
            # Recieve data from the server
            while len(data)<image_size:
                data += client_socket.recv(chunk_size)
            assert len(data) == image_size, "Image size is incorrect. %d, %d" % (len(data), image_size)
            return data
#==============================================================================
def main(addr):
    # Start PyGame:
    pygame.init()
    # Create a PyGame screen, and set its size to 640x480L
    screen = pygame.display.set_mode(image_res)
    # Set the window caption:
    pygame.display.set_caption("The Fishbowl")
    # Create a PyGame clock which will be used to limit the fps
clock = pygame.time.Clock()

timer = 0
previousImage = ""
image = ""

while 1:
    # Check if the exit button has been pressed:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

    # Timer to limit how many images requested from the server each second
    if timer < 1:
        image = get_frame(addr)
        # put more time on the clock
        timer = 5
    else:
        timer -= 1

    clock.tick()
    print clock.get_fps()

    screen.blit(string2Surface(image),(0,0))
    pygame.display.update()

if __name__ == "__main__":
    main((socket.gethostname(), 50000))
import socket
import pygame
import pygame.camera
from multiprocessing import Queue

image_res = (512, 384)

def serve(address, que):
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.bind(address)
        sock.listen(1)
        print "Your IP address is: ", socket.gethostbyname(socket.gethostname())

        # initialise the display window
        pygame.init()
        pygame.camera.init()
        cameras = pygame.camera.list_cameras()
        print cameras
        cam = pygame.camera.Camera(cameras[0], image_res)
        cam.start()

        print "Server Waiting for client on port", address[1]

        while 1:
            img = cam.get_image()
            img = pygame.transform.flip(img, 1, 0)
            client_socket, address = sock.accept()
            img = pygame.transform.scale(img, image_res)
            data = pygame.image.tostring(img, 'RGB')
            # Place frame on queue
            if que.empty():
                que.put(data)
            # Prepend the frame with the length of the data so that
            # the client knows how much data to receive
            data = str(len(data)) + '::' + data
            client_socket.sendall(data)
            #print "Frame size", len(data), "sent."
    except KeyboardInterrupt:
        print "Exiting..."
    finally:
        sock.close()
        print "Program Exited."

if __name__ == '__main__':
    address = ('127.0.1.1', 50000)
    serve(address, Queue(1))
#!/usr/bin/env python2

# Author: Raymond Barrett
# Email: barretrj@tcd.ie

import socket
import pygame
import pygame.camera
from multiprocessing import Queue

#image_res = (1040, 680)
#image_res = (1040, 680)
#image_res = (950, 540)
image_res = (512, 384)

def serve(address, que):
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.bind(address)
        sock.listen(1)
        print "Your IP address is: ", socket.gethostbyname(socket.gethostname())

        # initialise the display window
        pygame.init()
        pygame.camera.init()
        cameras = pygame.camera.list_cameras()
        print cameras
        cam = pygame.camera.Camera(cameras[0], image_res)
        cam.start()
        print "Server Waiting for client on port", address[1]

        while 1:
            img = cam.get_image()
            img = pygame.transform.flip(img, 1, 0)

            client_socket, address = sock.accept()
            img = pygame.transform.scale(img, image_res)
            data = pygame.image.tostring(img, 'RGB')
            # Place frame on queue
            if que.empty():
                que.put(data)
            # Prepend the frame with the length of the data so that
            # the client knows how much data to receive
            data = str(len(data)) + '::' + data
            client_socket.sendall(data)
            #print "Frame size", len(data), "sent."
        except KeyboardInterrupt:
            print "Exiting..."
        finally:
            sock.close()
            print "Program Exited."
    except ImportError:
        print "Import Error."

if __name__ == "__main__":
    address = ('127.0.1.1', 50000)
    serve(address, Queue(1))
#!/usr/bin/env python2

# Author: Raymond Barrett
# Email: barretrj@tcd.ie

import pygame
import json
from random import randint
from pygame.locals import *

SCREENSIZE = [1024, 768]
BOXSIZE = (300, 500)
BUBBLESIZE = (300, 86)
AVATARSIZE = (60, 60)
POS = (SCREENSIZE[0]/2 - BOXSIZE[0]/2, SCREENSIZE[1]/2 - BOXSIZE[1]/2)

class Staff(object):
    """docstring for Staff""
    pygame.font.init()
    myfont = pygame.font.SysFont("arial", 18)

    bg1 = pygame.image.load('Images/wordlebubble.png')
    bg1 = pygame.transform.scale(bg1, BOXSIZE)
    # bg2 = pygame.image.load('Images/WordleWordBox2.gif')
    # bg2 = pygame.transform.scale(bg2, (BOXSIZE[0]-10, BOXSIZE[1]-10))
    email = pygame.image.load("Images/email_button.png")
    email = pygame.transform.scale(email, (90, 40))

    pos = POS
    bobfactor = 1
    boboffmax = 3
    boboffmin = -3
    bobcount = 0

    def __init__(self, id):
        self.id = id
        self.avatar = pygame.image.load("UserInfo/Avatars/%s.png" % self.id)
        self.avatar = pygame.transform.scale(self.avatar, AVATARSIZE)
        self.usr = json.load(open("UserInfo/%s.json" % self.id, "rb"))

        self.label = self.myfont.render(self.usr['name'], 1, (0,0,0))
        self.name = self.myfont.render("Name: %s" % self.usr['name'], 1, (0,0,0))
        self.contact = self.myfont.render("Contact: %s" % self.usr['contact'], 1, (0,0,0))
        self.idlabel = self.myfont.render("ID: %s" % self.usr['id'], 1, (0,0,0))

        tags = self.usr['tags']
        self.tag_labels = [self.myfont.render(tag, 1, (0,0,0)) for tag in tags]
        self.tag = self.myfont.render("Working on: ", 1, (0,0,0))

        self.boboffset = randint(self.boboffmin+1, self.boboffmax-1)

    def display(self, screen):
        # display box
        if self.bobcount == 2:
            if self.boboffset >= self.boboffmax or self.boboffset <= self.boboffmin:
                self.bobfactor *= -1
            self.boboffset += self.bobfactor
        self.bobcount = 0

        email = pygame.image.load("Images/email_button.png")
        email = pygame.transform.scale(email, (90, 40))

        pos = POS
        bobfactor = 1
        boboffmax = 3
        boboffmin = -3
        bobcount = 0

        def __init__(self, id):
            self.id = id
            self.avatar = pygame.image.load("UserInfo/Avatars/%s.png" % self.id)
            self.avatar = pygame.transform.scale(self.avatar, AVATARSIZE)
            self.usr = json.load(open("UserInfo/%s.json" % self.id, "rb"))

            self.label = self.myfont.render(self.usr['name'], 1, (0,0,0))
            self.name = self.myfont.render("Name: %s" % self.usr['name'], 1, (0,0,0))
            self.contact = self.myfont.render("Contact: %s" % self.usr['contact'], 1, (0,0,0))
            self.idlabel = self.myfont.render("ID: %s" % self.usr['id'], 1, (0,0,0))

            tags = self.usr['tags']
            self.tag_labels = [self.myfont.render(tag, 1, (0,0,0)) for tag in tags]
            self.tag = self.myfont.render("Working on: ", 1, (0,0,0))

            self.boboffset = randint(self.boboffmin+1, self.boboffmax-1)

        #print self.boboffset

        def display(self, screen):
            # display box
            if self.bobcount == 2:
                if self.boboffset >= self.boboffmax or self.boboffset <= self.boboffmin:
                    self.bobfactor *= -1
                self.boboffset += self.bobfactor
            self.bobcount = 0

            email = pygame.image.load("Images/email_button.png")
            email = pygame.transform.scale(email, (90, 40))

            pos = POS
            bobfactor = 1
            boboffmax = 3
            boboffmin = -3
            bobcount = 0

        def __init__(self, id):
            self.id = id
            self.avatar = pygame.image.load("UserInfo/Avatars/%s.png" % self.id)
            self.avatar = pygame.transform.scale(self.avatar, AVATARSIZE)
            self.usr = json.load(open("UserInfo/%s.json" % self.id, "rb"))

            self.label = self.myfont.render(self.usr['name'], 1, (0,0,0))
            self.name = self.myfont.render("Name: %s" % self.usr['name'], 1, (0,0,0))
            self.contact = self.myfont.render("Contact: %s" % self.usr['contact'], 1, (0,0,0))
            self.idlabel = self.myfont.render("ID: %s" % self.usr['id'], 1, (0,0,0))

            tags = self.usr['tags']
            self.tag_labels = [self.myfont.render(tag, 1, (0,0,0)) for tag in tags]
            self.tag = self.myfont.render("Working on: ", 1, (0,0,0))

            self.boboffset = randint(self.boboffmin+1, self.boboffmax-1)

        #print self.boboffset

        def display(self, screen):
            # display box
            if self.bobcount == 2:
                if self.boboffset >= self.boboffmax or self.boboffset <= self.boboffmin:
                    self.bobfactor *= -1
                self.boboffset += self.bobfactor
            self.bobcount = 0

            email = pygame.image.load("Images/email_button.png")
            email = pygame.transform.scale(email, (90, 40))

            pos = POS
            bobfactor = 1
            boboffmax = 3
            boboffmin = -3
            bobcount = 0

        def __init__(self, id):
            self.id = id
            self.avatar = pygame.image.load("UserInfo/Avatars/%s.png" % self.id)
            self.avatar = pygame.transform.scale(self.avatar, AVATARSIZE)
            self.usr = json.load(open("UserInfo/%s.json" % self.id, "rb"))

            self.label = self.myfont.render(self.usr['name'], 1, (0,0,0))
            self.name = self.myfont.render("Name: %s" % self.usr['name'], 1, (0,0,0))
            self.contact = self.myfont.render("Contact: %s" % self.usr['contact'], 1, (0,0,0))
            self.idlabel = self.myfont.render("ID: %s" % self.usr['id'], 1, (0,0,0))

            tags = self.usr['tags']
            self.tag_labels = [self.myfont.render(tag, 1, (0,0,0)) for tag in tags]
            self.tag = self.myfont.render("Working on: ", 1, (0,0,0))

            self.boboffset = randint(self.boboffmin+1, self.boboffmax-1)

        #print self.boboffset

        def display(self, screen):
            # display box
            if self.bobcount == 2:
                if self.boboffset >= self.boboffmax or self.boboffset <= self.boboffmin:
                    self.bobfactor *= -1
                self.boboffset += self.bobfactor
            self.bobcount = 0

            email = pygame.image.load("Images/email_button.png")
            email = pygame.transform.scale(email, (90, 40))

            pos = POS
            bobfactor = 1
            boboffmax = 3
            boboffmin = -3
            bobcount = 0

        def __init__(self, id):
            self.id = id
            self.avatar = pygame.image.load("UserInfo/Avatars/%s.png" % self.id)
            self.avatar = pygame.transform.scale(self.avatar, AVATARSIZE)
            self.usr = json.load(open("UserInfo/%s.json" % self.id, "rb"))

            self.label = self.myfont.render(self.usr['name'], 1, (0,0,0))
            self.name = self.myfont.render("Name: %s" % self.usr['name'], 1, (0,0,0))
            self.contact = self.myfont.render("Contact: %s" % self.usr['contact'], 1, (0,0,0))
            self.idlabel = self.myfont.render("ID: %s" % self.usr['id'], 1, (0,0,0))

            tags = self.usr['tags']
            self.tag_labels = [self.myfont.render(tag, 1, (0,0,0)) for tag in tags]
            self.tag = self.myfont.render("Working on: ", 1, (0,0,0))

            self.boboffset = randint(self.boboffmin+1, self.boboffmax-1)

        #print self.boboffset

        def display(self, screen):
            # display box
            if self.bobcount == 2:
                if self.boboffset >= self.boboffmax or self.boboffset <= self.boboffmin:
                    self.bobfactor *= -1
                self.boboffset += self.bobfactor
            self.bobcount = 0

            email = pygame.image.load("Images/email_button.png")
            email = pygame.transform.scale(email, (90, 40))

            pos = POS
            bobfactor = 1
            boboffmax = 3
            boboffmin = -3
            bobcount = 0
self.bobcount += 1

p = (self.pos[0], self.pos[1] + self.boboffset)

screen.blit(self.bg1, p)
# screen.blit(self.bg2, (self.pos[0]+ 5, self.pos[1] + 5))

avpos = (p[0] + BOXSIZE[0]/2 - AVATARSIZE[0]/2, p[1]+AVATARSIZE[1]/2)
# Draw avatar
screen.blit(self.avatar, avpos)
# Draw name
screen.blit(self.name, (self.pos[0] + 30, self.pos[1] + avpos[1]))
cnt = 0
x = self.pos[0] + 30 + self.tag.get_width()
for tag in self.tag_labels:
    print x + tag.get_width()
    if x + tag.get_width() > SCREENSIZE[0]/2 + BOXSIZE[0]/2 - 30:
        y += 30
        x = self.pos[0] + 30 + self.tag.get_width()

    screen.blit(tag, (x,y))
    x += tag.get_width() + 5
cnt += 1


#pygame.display.update()

def get_avatar(self):
    return self.avatar

def get_name(self):
    return self.usr['name']

def set_pos(self, pos):
    self.position = pos

def get_pos(self):
    return self.position

def bob(self):
    if self.bobcount == 2:
        if self.boboffset >= self.boboffmax or self.boboffset <= self.boboffmin:
            self.bobfactor *= -1
            self.position = (self.position[0], self.position[1] + self.bobfactor)
            self.boboffset += self.bobfactor
            self.bobcount = 0
    else:
        self.bobcount += 1

class WordleWord():
    """WordleWord: Takes in a word, looks in JSON file for matching word, Reads in information for display."""
pygame.font.init()
header = pygame.font.SysFont("arial", 30)
myfont = pygame.font.SysFont("arial", 15)

#screen = pygame.display.set_mode(SCREENSIZE)
#pygame.display.set_caption('Wordle')

bg1 = pygame.image.load('Images/wordlebubble.png')
bg1 = pygame.transform.scale(bg1, BUBBLESIZE)
# bg2 = pygame.image.load('Images/WordleWordBox2.gif')
# bg2 = pygame.transform.scale(bg2, (BOXSIZE[0]-10, BOXSIZE[1]-10))

pos = POS
position = pos

bobfactor = 1
boboffmax = 3
boboffmin = -3
bobcount = 0

def __init__(self, w):
    for f in json.load(open('WordleWord.json', 'r')):
        if f['id'] == w:
            self.word = f['word']
            assoc = f['associated']
    # Read in relevant information from JSON file
    self.associated = [Staff(ass) for ass in assoc]
    for i in xrange(len(self.associated)):
        AvatarPos = (self.pos[0] + AVATARSIZE[0],
                     self.pos[1] +70+ AVATARSIZE[1]/2 +
                     ((AVATARSIZE[1]+30)*i))
        self.associated[i].set_pos(AvatarPos)

        self.label = self.header.render(self.word, 1, (0,0,0))
        self.boboffset = randint(self.boboffmin+1, self.boboffmax-1)

def display(self, screen=None):
    # display box
    # print self.word
    if self.bobcount == 2:
        if self.boboffset >= self.boboffmax or self.boboffset <= self.boboffmin:
            self.bobfactor *= -1
            self.boboffset += self.bobfactor
            self.bobcount = 0
        else:
            self.bobcount += 1

    p = (self.pos[0], self.pos[1] + self.boboffset)

    screen.blit(self.bg1, p)
    # screen.blit(self.bg2, (self.pos[0]+5, self.pos[1] + 5))
    screen.blit(self.label, (p[0] + BUBBLESIZE[0]/2 - self.label.get_width()/2, p[1] + 25))
    # display avatars
    # print self.associated
    for ass in self.associated:
        ass.bob()
        AvatarPos = ass.get_pos()
        Avatar = ass.get_avatar()
        x_pos, y_pos = self.pos
        # Put in bubble
        screen.blit(self.bg1, (x_pos, AvatarPos[1] + Avatar.get_height()/2 - BUBBLESIZE[1]/2))
        # Put in Avatar
        screen.blit(Avatar, AvatarPos)
        # Put in name
def check_click(self, point):
    i = 0
    found = -1
    for ass in self.associated:
        p = ass.get_pos()
        # If avatar xpos < point xpos < avatar xpos + get_width
        if p[0] > point[0] and point[0] < p[0] + AVATARSIZE[0] + ass.label.get_width() + 30:
            # If avatar ypos < point ypos < avatar ypos + height
                found = i
        i += 1
    return found

def change_state(self, state, eventpos):
    new_state = state
    av = -1
    if state == 'tank':
        new_state = 'word'
    elif state == 'word':
        if av >= 0:
            #Change state
            new_state = 'user'
        else:
            # Else close screen
            new_state = 'tank'
    return (new_state, av)

def main(self):
    pygame.init()
    state = "tank"
    showWord = False
    showUser = False
    while True: #Game Loop - Detects Key Presses and Mouse Clicks
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.display.quit()
                exit()
            elif event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:   #Left Click
                    state, av = self.change_state(state, event)
                elif event.button == 3:
                    pass
            elif event.type == pygame.MOUSEBUTTONUP:
                if event.button == 1:
                    pass
                elif event.button == 3:
                    pass
            if state == "word":
                self.display()
            elif state == "user":
                self.associated[av].display(self.screen)
if __name__ == '__main__':
s = {
    'id': 'word2',
    'word': "Hana",
    "associated": ['Ray', 'Mark', 'Eimear', 'Peter']}

json.dump(s, open('WordleWord.json', 'wb'))

a = WordleWord("word2")
a.main()