# Human Harvesters

***Interim Report***

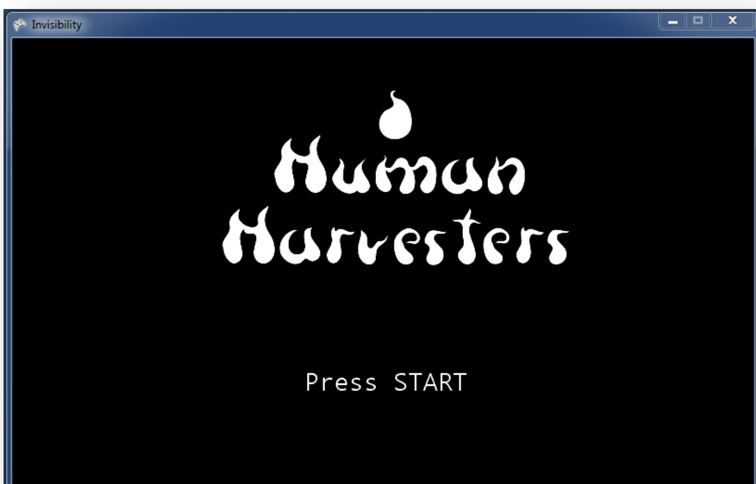
## Current Progress

### Development schedule

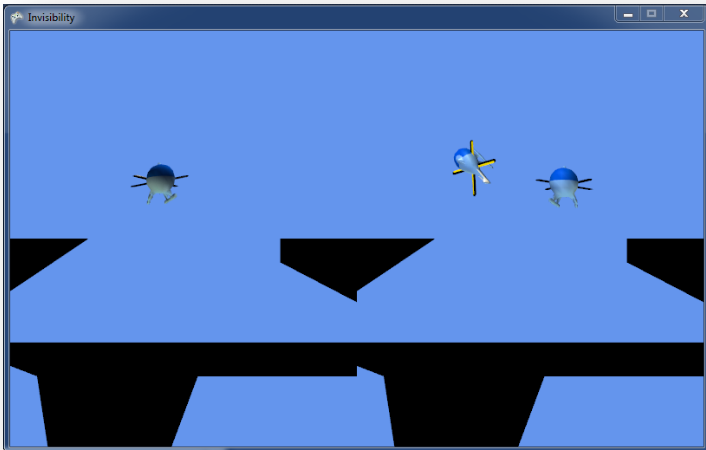
| Task | Planned completion date | Status |
|---|---|---|
| Physical prototype | 20 March | |
| Tutorials and research | 20 March | |
| **Functional minimum** | | |
| Player has an Avatar that can move | 20 March | |
| Player can kill other players | 27 March | |
| Player re-spawns when dead | 3 April | |
| Map exists | 27 March | |
| Ambient light | 3 April | |
| **Low target** | | |
| Map has obstacles | 3 April | |
| Light mechanism for invisibility with time limit in place | 3 April | |
| Basic shading/lighting (deferred rendering) | 17 April | |
| Humans are on the map | 3 April | |
| Players can consume humans | 10 April | |

| Desirable target | | |
|---|---|---|
| Interim report and demo | 17 April | |
| Texturing | 17 April | |
| Game menu / GUI etc. | 10 April | |
| Basic sounds | 24 April | |
| Animated avatars | 17 April | |
| Humans have actual models | 17 April | |
| Background music (not necessarily self-composed) | 24 April | |

## Annotated screenshots

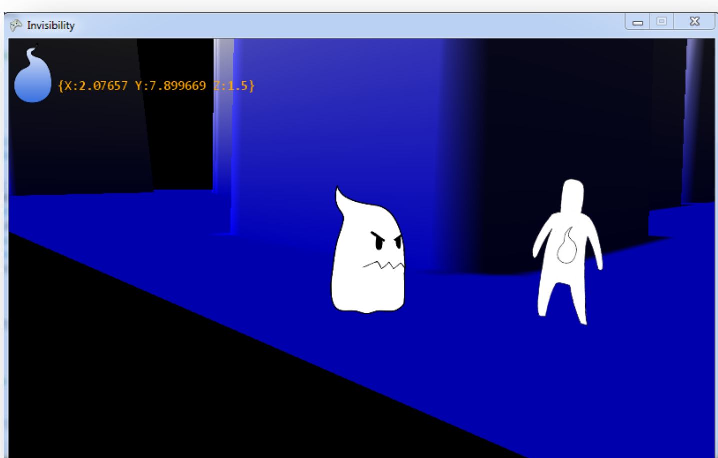



Our menu screen – pressing 'start' ('enter' on keyboard) brings you into a new multiplayer game. We've implemented our game with the idea of mode-switching to transition from mapping input to menu-selections or game controls. Our assets are loaded at start-up to allow for lightning-fast transitions between modes.




Early test phase – we used provided models from tutorials for testing purposes. Here you can see our four-player split-screen view, the first two players having moved around. We allowed for keyboards and controllers to work together to allow for a seamless transition from PC to XBOX.


Here you can see further development in our early-phase (still using borrowed models) – we have two players moving separately, and the "humans" (ships) are spawned randomly along the play plane at the start of the game.




This is after our second architecture re-write for integrating the map – we have players that can move and humans are on the map. However, interactions and physics (collisions) are still missing.

# Implementation Challenges

## Architecture

The key aspect of our game design that has taken the most time, energy, pain, anger, fear, and tears is the architecture of how our game will be structured.

Initially, we all began coding in separate projects, experimenting with pieces of the game that we were to develop and hoping to merge things together eventually.

We then realized that for a game of any scope to function properly and be workable, we needed to devise an architecture that was extensible, formal, and flexible for the addition of new components. Thus, a team of our group sat down for upwards of eight or ten hours over a weekend to discuss and plan the master architecture for the system. We researched typical recommended XNA game architectures online, read about C# features and how they were integrated into the XNA framework, and attempted to utilize XNA's "guideline" architecture features (such as registering game components and providing service interfaces). Over the next few days, we implemented the skeleton of this architecture, and were just beginning to flush it out with our "actual game" when we showed the "game" to the class.

At this point, though we did not have much functioning, we were determined that we would be able to build upon the solid architecture that we had created and whip up our game in no time. Don't laugh.

Of course, there were problems. Here are some of the main ones:

- **We did not balance architectural 'style' with ease of engineering.** Making a codebase that is beautiful architecturally, and making a codebase that is easy and fun to build onto, are two different things. They of course have an area of intersection, but in our initial design fell far more into the 'architecturally beautiful' side of the spectrum than the 'ease of engineering.'

- **We crafted an entire architecture before having made a game.** We knew that we were doing this; after all, we thought that if we planned enough at first, and if we planned well enough, we would be able to avoid architectural overhauls down the road. However, we just could not predict how the pieces of the game needed to interact, and our plan lead to code that was difficult to write and somewhat cumbersome to use in order to strictly maintain the architectural values.

- **We did not coordinate further architecture changes once we went off on our own branches.** After the initial architecture was made, different groups and individuals on the team went off to craft their own sections, and some found it necessary to make changes to how the architecture worked. This may have been inevitable, but we failed to properly coordinate how these changes were happening team-wide. The result was an absolute mess of code integration and modification (i.e. throwing a lot away) once we needed to starting bringing things together.

## Key Lessons Learned So Far

1. **We need to all work on the same code base.** We must emphasize the celebrated 'git workflow' where one checks out a branch, makes some changes and tests them, and then immediately merges back in. We continually have many versions of the game going on at once, and there is no consensus as to which is the 'main' one. Having parallel uncoordinated development go on for too long leads to work being thrown away.

2. **We need to have hard deadlines and stick to them.** Because of the lack of coordination about architecture issues, part of our group would re-implement pieces of another architecture to make their new one 'catch-up' to where it had been. This lead to multiple people in the group working on the same thing at the same time, and because of this, deadlines were ignored because of grand changes and merging issues. When the 'real' deadlines came near, we suddenly looked again at our plan and realized that all of the pieces we had been hoping to put together were at unclear stages of completion because different groups within the team had gone on their own architecture and own track.

3. **We need to have tighter communication.** We currently have many different places to look for messages and issues: stypi_ (a real-time document collaboration tool), Google Docs, Trac, a forum, email, the codebase, the repository of documents, the Twiki, Skype, texts, calls… This all leads to scattered information and loose guidelines about what is being done by whom and what the status of the project is. We need to pick one unified place to put our important information and stick to it.