**Project Analysis Document – Joseph Henry**

**Program:**

Network Jumper is a side scrolling game in which the player has to jump onto platforms for as far as possible without falling into gaps or before the screen has scrolled past the player. The platforms will be placed on the screen with procedurally generated heights. This game is designed for a younger audience.

**Why the task suited to computational solutions:**

The program will require an algorithm to control the peer to peer data transfers to allow the multiplayer function to work. The gravity feature will also need another algorithm to change the direction of the player’s character while jumping e.g. go up and back down at the other end or fall when going off an edge. Furthermore, the program will need some code to generate the obstacles into the course to allow the course to be infinitely long map.

**Stakeholders:**

Joseph Hurr

Victor Che

James Lowe

Ethan Cox

These people will test my game for me at each prototyping stage and give feedback on alterations I could make to improve the function of my game.

**Stakeholder Meeting Minutes:**

Date – 14.10.19

Attendees – Joseph Henry, Joseph Hurr, Victor Che, James Lowe, Ethan Cox

Agenda – Explain concept projects to rest of the group, then discussed ideas for possible improvements to our projects

Feedback Received – Rather than procedural generation, use an RNG to pick from a list of pre-set options in a list.

Final Thoughts – I agree that RNG could be a better way of generating the level rather than making the game completely procedural. It would still mix up the level layouts but be much easier and quicker to implement reducing many problems that could possibly come up from complete procedural generation.

**Existing Solutions:**

Flappy bird –Flappy bird is a side-scroller mobile game where the player controls a bird attempting to fly in-between two green pipes without touching them.

The game scrolls sideways at continuous speed and has a gravity function. The map is procedurally generates gaps for the bird to be navigated through.

Jetpack joyride – Jetpack Joyride is another mobile side-scroller in which the player controls a man who is continuously moving to the right in an attempt to cause as much havoc in this continuous hallway as possible.

The game scrolls sideways at increasing speed and has a gravity function. It also has a procedurally generated course for the player to navigate through, with objects such as lasers, which force the player to change position or lose the game.

Super meat boy –In the game, the player controls Meat Boy, a red, cube-shaped character, as he attempts to rescue his girlfriend, Bandage Girl, from the game's antagonist Dr. Fetus.

The game scrolls with the player and has a gravity function and the ability to slow down a players fall by sticking to the wall. Super meat boy also has set map layouts.

I like the idea of a base of a continuous scrolling speed but if the player is too far ahead then the scroll should be centred on the leader to prevent players hitting the edge of the screen. I also believe a basic gravity feature would fulfil the purpose needed for my game to become functional. I would like to implement a similar course generation as seen in jetpack joyride and flappy bird. A different assortment of obstacles each time the game is started. Personally, I do not think the set map layout is a good implementation to add to my game, as the game would lose its replay ability thus making it more unenjoyable for the players to play over time.

**Essential Features:**

Multiplayer for multiple clients to play against each other in real time.

Multiplayer will be in sync across platforms.

Random generation for platform heights and gaps to add variety to the game rather than repetition of the same obstacle.

Gravity for players to be able to jump in a realistic manner to get onto the different height platforms.

There will be controls to allow the player to move.

**Limitations:**

Latency in Peer to peer games tends to be higher than server-client games. This could cause user experience to suffer. Each peer must wait for every other peer's packet before simulating the next network frame, resulting in all players experiencing the same latency as the player with the worst connection.

It is very difficult to keep all peers synchronised. Minute differences between peers can escalate over time to game-breaking paradoxes.

Each peer must communicate with all other clients, limiting the number of connected players to prevent the game from running slow.

**Prototypes:**

Prototype 1: Create a basic connection between clients to build the multiplayer feature. Allow two players to connect to the main host. Have all clients synced with each other and minimise lag.

Prototype 2: The second prototype will have allow the player to move their character to move left and right as well as jump up with the character falling back to the ground after. The ground should be a flat surface in which the character can stand on without falling through the floor. It will also include the feature of the screen scrolling to the right at a consistent speed.

Prototype 3: Maps will now be generated with gaps and platforms in them to provide some difficulty for the player to run through. Some platforms will also move up and down to create a new obstacle for players to encounter. When a character falls into a gap or moves off the viewable area the player will be turned into a spectator of the rest of the players until the last player is standing.

**Success Criteria:**

Prototype 1 –

* The program will allow for 2 players to connect to each other through sockets
* The multiplayer will have all clients in sync with a stable connection.
* Controls will WASD. These controls will affect the position of the client’s character.
* Test to see if movements are synchronised on both screens.

Prototype 2 –

* Collision physics implemented to the player and the environment.
* Gravity implemented into the game to allow a jump rather than moving up and down.

Prototype 3 –

* Gaps will be generated into the terrain either side of platforms.
* Platforms will be put into the terrain at random intervals with heights being procedurally generated.
* Moving platforms will also be implemented which will constantly move up and down at a set speed.
* When characters fall off the screen will be turned into a spectator of the ongoing match.
* Characters which go off the back of the play area will also be turned into a spectator.

**Minimum System Requirements**

* Processors: Intel Atom® processor or Intel® Core™ i3 processor
* Disk space: 1 GB
* Operating systems: Windows\* 7 or later, macOS, and Linux

**Software Requirements**

* Python with the pygame extension installed.
* Podsixnet installed to allow for a socket based connection to