STANDALONE

BLAIR CROSS ADV.H COMPUTING SCIENCE PROJECT
SCN 061992219

Project Plan.

Project Proposal.

Game Summary.

You play as an agent from a cyberware firm, in a first person position. You have been tasked with infiltrating a building, ~~finding the server with the stolen data on it and destroying it~~ 12/3/18 finding an ID for a targeted file and searching it up on a data server. You must then escape.

Core Mechanics.

- ~~Two approaches (lethal and nonlethal).~~
- Stealth based, slow progressive nonlinear gameplay.
- Crouch, walk, run, jump.
- ~~Inventory for weapons and key items.~~ 13/11/17 Not necessary / too complex for timescale.
- Hackable locks (Some form of minigame similar to Mankind:divided). 13/11/17 Note that this is now only for the accessing the data on the server.
- Patrolling enemies.
- ~~11/9/17 Third person.~~ 16/11/17 Too complex, opted for a 1st person view.
- ~~11/9/17 Hiding in shadows makes you less detectable.~~ 13/11/17 Too complex and not needed.

Gameplay.

Sneak or shoot your way through a floor of a building. Get to the server and destroy 27/3/18 retrieve the data, either physically or digitally - each with their own consequence - then escape the building, all while staying alive and undetected.

Music.

Unimportant. Blend between pressured and relaxed. Inspired by old games (to keep effort low).

Art Style.

Low poly, cyberpunk minimalist. Like Mirror's Edge but grimey, and far more futuristic and dystopian. Inspiration from Akira and Ghost in the Shell (1995).

Appropriate level of complexity?

'an interface appropriate for your users that validates all inputs'	The UI for the game will have a menu system (just buttons) but more importantly the ~~options screen will allow players to change game settings like sound volume that will be constrained to a range of 0-100. Also it will be possible for users to enter keycodes ingame which will have to be validated against the actual keycode.~~ 12/3/18 Finding the file requires the player to type in an 'ID'. This ID is validated to make sure it take s the correct format. The UI will be designed for use with a desktop monitor (16:9).
'interfacing with stored data'	~~The game will read dialogue lines from a text file and put them in a queue, ready to be displayed to the screen one by one.~~ 12/3/18 All the file 'IDs' will be generated by a separate program and imported into Unity as a text while which will get read in and displayed as a list of files.
'2-D arrays, arrays of records or linked lists'	The game will generate a 'map' of the server network to be hacked and store it as a 2d array. Which will then be displayed to the screen.
'a binary search, a sort algorithm or other coding of similar complexity'	Looking through a computer to find a ID will require the player to sort or search a database in order to find it.

~~'Recursion'~~
 	
12/3/18 Removed because it didn't fit the architecture I made.
	~~Recursion will be used in the enemies patrolling AI. i.e void Patrol () {
	//Move to next waypoint
	If (! playerDetected) {
	Patrol();
	}
	}~~

12/8/18 'Queues'	The dialogue system uses a queue to store lines of dialogue waiting to be read. The dialogue system was reintroduced as I had the time to do it, and the player required some kind of direction to play the game.

The project will allow me to apply planning (how and when I will tackle a mechanic), research and analysis (how other games handle stealth and how they did it), problem solving and evaluation. With strict discipline and proper project management (I will be using something similar to Trello to stay on task) the project will be completed in a couple of months. I will also use source control to make sure I have a backlog of every single change I make to the project. There are no potential barriers, as the engine and modelling software I will use is free. Any knowledge I lack can be gained on the internet or for a £20~ cost (for a book). I have over a years experience with Unity (the engine) and C#. I have already made several games through to completion. All of the mechanics I intend to include have already been done before, and many have lots of detail on the internet about them. The stealth mechanic which has no

particular attachment to the assignment is something I've done previously, and can just import it to save time.

Research Documents.

What makes an A.I? Document detailing research done on A.I.

Sources: Programming Game AI by Example (Book) written by Mat Buckland.

I've found that there are two categories in the subject: Academic AI and Game AI. Academic AI focuses on creating a realistic response from a situation, in order to mimic human emotion. However, Academic AI doesn't doesn't care about processing time. An Academic AI is designed to run on servers with mountains of processing power and can take day or even month to reach a conclusion. Game AI on the other hand can come to a decision in a single frame, because its logic is *if this do that*.

AI is a large topic to try and cover, so going from a beginner to expert in the time given is unrealistic. So I have decided to choose an easier type of structure: A finite state machine. A finite state machines that the guards have a set amount of states. States like `LookForPlayer` and `PatrolWaypoints`. These states have behaviours written inside them that control the agent - agent being anything that is moved by its own AI. On the guard itself it will have a script detailing when to change behaviours. A FSM will only ever be in one state at a time. The FSM described in the book has a method for changing states, which allows states to have an enter, update and exit action. For instance, when a guard detects the player, it would change state to `PursuePlayer`. The enter method for `PursuePlayer` would perhaps be to make the guard say "Hey!" and take their gun out. The update method would be to chase and shoot the player, and the exit would be called when the guard changes state and would make it say "Blast he (or she) got away!" and put their gun away.

An alternate approach to the FSM is to get rid of the 'chip in the brain' that determines when to switch behaviours, and put the rules in the behaviours themselves. So that means that its the class behavior `PursuePlayer` that determines when to revert back to patrolling, instead of a higher up managing script.

What makes a stealth game? Document detailing research findings.

"A **stealth game** is a type of video game that tasks the player with using *stealth* to avoid or overcome antagonists." - https://en.wikipedia.org/wiki/Stealth_game

Unlike most games, stealth games seem to vary in gameplay by a lot. From being top down to a first person perspective, from allowing the player to kill to awarding a non-lethal play. Level design is more important than in other games, as factors like cover, light and guard patrolling/positioning greatly effect the enjoyment from the game. Players prefer a level made so that it has many approaches.

Key Points.

Some key points that I've found players find a necessity;

- The best stealth games give players multiple paths to take for every play style (lethal/nonlethal). This is a feature of game design and I will iteratively design a single level against feedback from the beta testers.
- Stealth games about player mobility, allowing the player to sneak through vents/along railings etc. This is a feature of game design and will be treated like the point above. Good mobility means the player spends less time bored, wandering around paths.
- Stealth is the key mechanic of a traditional stealth game. Every other mechanic should enhance the player's ability to avoid or take down enemies covertly. This is a mechanic feature and I will decide on what mechanics I will use later on.
- Traditionally, combat should be a last choice. But in more modern games it's given as a player choice. This is a game design choice on how I allow/reward the player for their approach. To keep my list of tasks manageable, I feel like I will stick to the traditional choice and not turn my game into an FPS.
- RPG elements allow the player to truly play their way and let them hone their skills. For the sake of keeping my project simple, I will ignore this.
- Use of environment should allow the player to out maneuver guards. This is a feature from level design and shall be treated like the first two points.
- "Puzzling should increase the player's awareness and mobility."[1] A puzzle section is a key part of fulfilling the programming requirements of the task, so I will definitely feature this in the game.

In order to further my knowledge on what makes a stealth game good, I've compiled a list of 'classics' that I will play in my spare time. These treasured by fans of the genre and are generally critically acclaimed by everyone. I've also chosen to go for classics because aesthetically, they aren't impressive anymore, and that will allow me to spend less time on asset creation if everything is a low-poly. I've also restricted the list to 3 games because I'm not made of time.

- Deus Ex (2000)
    - Result: Deus Ex emphasizes what choices the player makes by calling them out at the end of the mission. I.e. leave all the guards alive and a soldier will walk past and complain about you making their job harder. The shooting model felt off, and in general it was boring to play as the level design was uninspired.
- Metal Gear Solid 2 (2001)
    - Result: MGS2 is a great game gameplay wise. Its camera perspective is third person and top down, allowing for more awareness of the situation and making you feel like a real spy. Although it's major flaw are the invasive cinematics. It's so focused on telling a story that there is about 30 minutes of reading poorly written, mostly meaningless text. This keeps gameplay sparse and eventually makes the player irritated every time their comms beep and they have to wait while some self-explanatory mechanic is explained to them for 10 minutes.
- Thief: The Dark Project (1998)

Questions to ask:

- Position of camera? (top down / first person / third person)?
- Most important gameplay mechanic? (Hiding in shadows / sound detail / mobility)?
- Prefered approach to a situation? (Lethal takedowns / non lethal takedowns / outmaneuver completely)?

Sources:

https://www.youtube.com/watch?v=U9BUaDN36a0

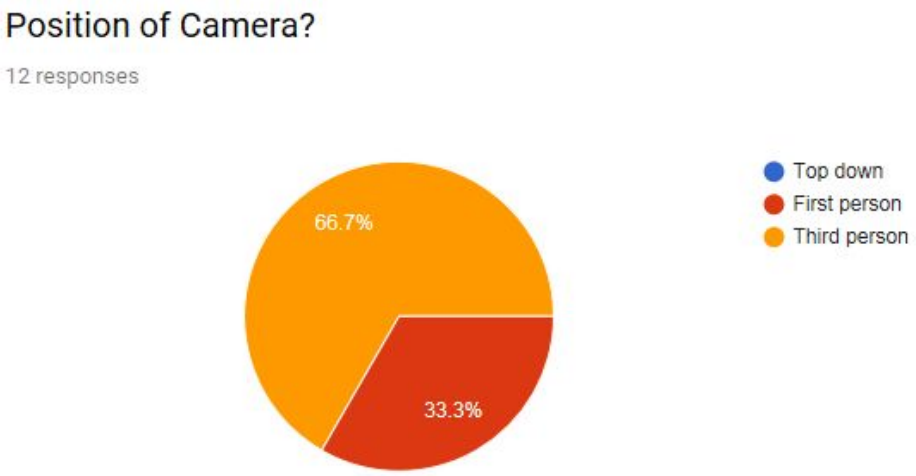
https://en.wikipedia.org/wiki/Stealth_game [1]

Selecting strategies.

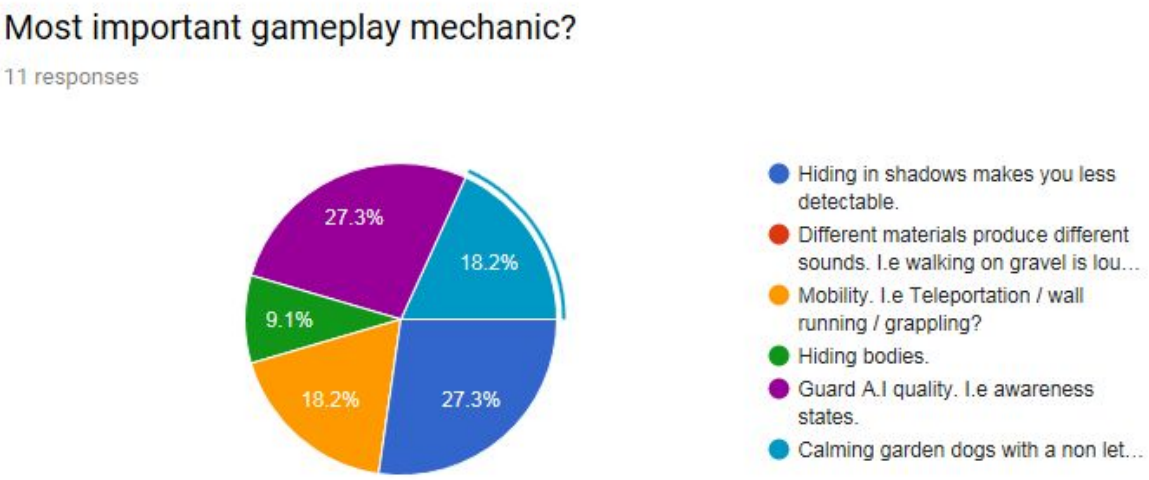
Unity and C#	**Unreal and C++**	**Cryengine and C++**
• Previous experience with engine. • Already know C#. • Extensive tutorials and documentation online. • Powerful graphics.	• Don't know C++. • Never used engine before. • Little amount of tutorials online. • Very powerful graphics.	• Don't know C++ • Never used engine before. • Some tutorials. • Very powerful graphics. • No asset store.

I have chosen to use Unity and C# to make my project, as I already have experience with both of them. It's designed towards making games, as it's a game engine. It supports real time and baked lighting, meaning my project can run on lower spec machines.

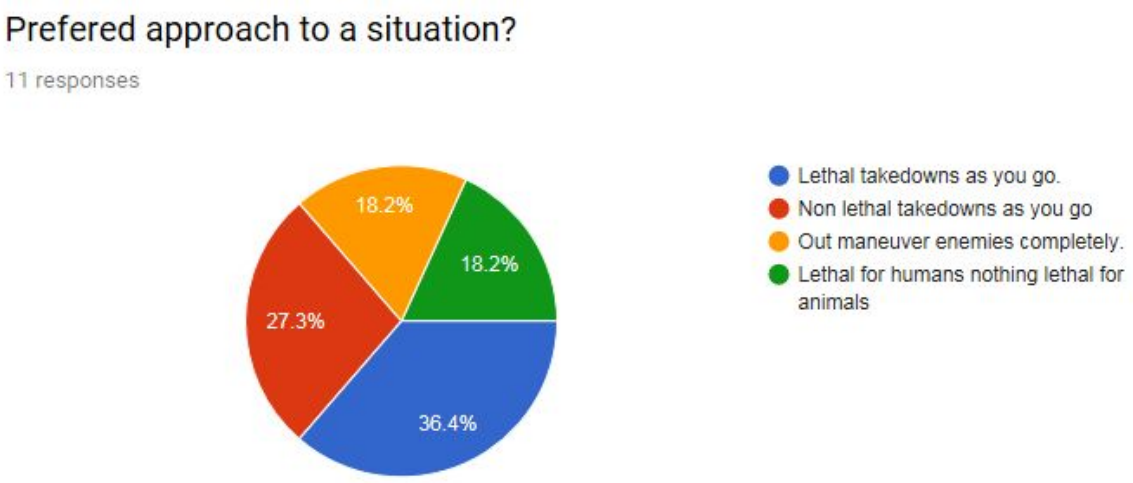
User survey results.



Third person was mostly prefered. But because a 3rd person camera is extremely difficult to implement and get right I have overridden the choice and went with a first person camera.



This was more of a mixed batch. The two most chosen choices were hiding in shadows and guard AI quality. I can imagine that detecting when the player in in shadow will be difficult so it will be ignored. Guard AI will have multiple states now.



People seem to prefer being lethal, and others are animal sympathisers. 12/1/18 I will keep the takedowns lethal, as I no longer intend to implement different weapons into the game due to time restrictions.

Feasibility Study.

Economics;
There will be no costs for the project. All software is free but a book on AI is require, which is £10 and affordable. Overall the project is economically feasible.

Legal;
The game won't collect or store users data. The game is free and, providing all audio and images I use are under the creative commons act, then there should be no legal issues.

Time;
The program will take 7 months to develop. It is hard to say how many hours, but I will be using my spare time. The project *should* be feasible in terms of time.

Technical;
My home computer, the one I will be developing on, has a GTX 1070 and an AMD fx850 which is more than beefy enough to handle game development. There should be no problems with technical issues.

"Unity is royalty free. We don't charge on a per title basis or require a revenue share model." - https://unity3d.com/unity/faq

"Any creation you make as an artist with Blender is your sole property, and can be applied for any purpose you choose to." - https://www.blender.org/support/faq/

~~"Free, fully featured IDE for students, open source and individual developers" - https://www.visualstudio.com/vs/~~
23/3/18 "VS Code is free for private or commercial use." - https://code.visualstudio.com/docs/supporting/faq

Basic Project Plan.

The major task is:

Make a stealth game

Basic outline of major tasks:

Sub Task	Time Allowed (hrs)	Target Date

Analysis	7	11/09/17
Design	3	13/10/17
Implementation	50	26/02/18
Testing	10	09/04/18
Documentation	10	27/04/18
Evaluation	3	23/04/18

Tasks and sub-tasks are tracked on an online tool called ‘Hacknplan’ which I abbreviate to HAP. HAP is designed to be an agile tool specially for the development of games. Each milestone has an array of tasks which can be dynamically rescheduled. Each task can have multiple subtasks. It also provides a timeline and allows me to log each session with a description and the amount of time I spent on it. Tasks can store data like a due date, estimated time cost, current time spent etc. Actual changes and additions/deletions to the project will be tracked on github.

Detailed Project Plan.

Sub Tasks

Analysis tasks:

- Find out what makes a stealth game enjoyable. (3hrs - 5hrs)
- Find out what design choices people prefer via survey. (1hr - 2hrs)
- Research how to create basic AI in a 3D environment. (Over several months)
- Effective and efficient OOP practices. (Over several days)

Project Plan tasks:

- Create this document. (3hr - 8 hrs)

Requirements Spec tasks:

- Determine requirements for user. (1hr)
- Determine functional and nonfunctional requirements. (1hr)

Test Plan Tasks:

- Create test plan against the requirement spec. (3hrs- 4hrs)

Design and Develop tasks:

- Design:
    - User interface. (2hrs - 4hrs)
        - Splash screen.
        - Main menu.
        - Game over menu.
        - Options menu.
        - Pause menu.
        - Some form of status panel.
    - Flowcharts. (5hrs - 10hrs)
        - Player states and laws including anims.
        - Behaviors for objects.
    - Finite state machine for guards.
- Implementation:
    - Dialogue and its system. (3hrs - 7hrs)
        - Write dialog (it won't be good).
        - Event system for controlling when and how the dialogue is displayed.
    - Gameplay flow. (1hr)
        - How the player will move through the level and how many approaches they get.
    - AI state machine and behaviours. (2hrs)
        - Why the guards react and how.
        - Design the individual behaviours (How the actually pursue or evade etc.).
    - The hacking puzzle. (10hrs - 15hrs)
        - The actual mechanics.
        - How it will read in the data and in what form. (Text file / picture?).
    - Interactions between objects. (2hrs - 4hrs)
        - Player interacting with object.
        - Game manager perhaps.
        - Player / Guard shooting objects / each other.
    - Assets. (15hrs - 25hrs)
        - Wall Piece.
        - Pillar Piece.
        - Door Piece.
        - Vent Piece (and it's varieties).
        - Pistol and Automatic rifle.
        - Player (With basic walk anims).
        - Guard (With basic walk and shoot anims).
        - Background Pieces (Squares with windows basically to sit behind the level).

Final Testing tasks:

- Component Testing will be done throughout the implementation stage.
- Acceptance testing. (7hrs)

Evaluation tasks:

- Evaluate whether or not it's fun. (10min)
- Evaluate; (4hrs)
    - Robustness.
    - Reliability.
    - Portability.
    - Efficiency.
    - Maintainability.
    - Overall management of project.

Time Allocation.

Analysis

Sub Task	**Time**	**Target Date**	**Completed**
Project proposal	20mins	08/09	11/09/17
Research documents (AI)	2hrs	09/09/17	09/09/17
Research documents (Gameplay)	2hrs	10/09/17	09/09/17
Requirements spec	2hrs	10/09/17	10/09/17
Scope and Boundaries	2hrs	11/09/17	10/09/17

Research

Sub Task	**Time**	**Target Date**	**Completed**
Stealth games research	2hrs	17/09/17	17/09/17
AI research	20hrs	12/10/17	12/10/17
Effective OOP programming patterns for game design.	2hrs	17/09/17	17/09/17

Design

Sub Task	Time	Target Date	Completed
UI design	2hrs	10/10/17	09/10/17
Class Diagrams for systems	3hrs	13/10/17	13/10/17
Test plan	5hrs	13/10/17	13/10/17

Implementation

Sub Task	Time	Target Date	Completed
Create and structure UI	10hrs	20/02/18	20/02/18
Create AI	13hrs	20/02/18	20/02/18
Create character controller	3hrs	01/02/18	01/02/18
Create character motor	4hrs	01/02/18	01/02/18
Create Gameplay objects and their interactions.	30hrs	26/02/18	26/02/18
Asset creation (models and stuff)	20hrs	26/02/18	26/02/18

Testing

Sub Task	Time	Target Date	Completed
Beta testing	1hr	01/04/18	01/04/18
Systematic testing	3hrs	01/04/18	03/04/18
Component testing	3hrs	09/04/18	09/04/18

Documentation

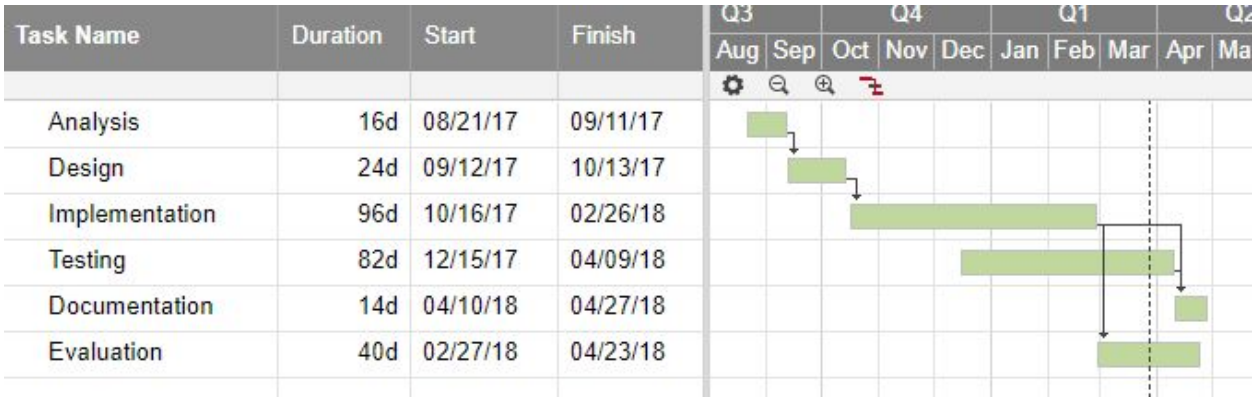
Sub Task	Time	Target Date	Completed
Generate user guide	20mins	27/04/18	27/04/18

Evaluation

Sub Task	Time	Target Date	Completed

Evaluate whether its fun	10mins	20/04/18	21/04/18
Robustness	10mins	23/04/18	23/04/18
Reliability	10mins	20/04/18	22/04/18
Portability	10mins	20/04/18	20/04/18
Maintainability	10mins	23/04/18	21/04/18

Gantt Chart



Resources to be used.

- Computer with Windows 10.
- Unity ~~v5~~ 12/1/18 Upgraded to v2018, just because.
- Computer with **a** graphics card.
- ~~Visual studio 2017.~~ 15/1/18 Upgraded to VS code as it's faster and more suited to what I need it for.
- Remote repository for version control (gitHub will be used).
- Blender (for modeling).
- Google drive for storing files.
- 12/9/18 yEd for making diagrams. I added this because every other diagram software costs money, but yEd is free.

Requirements Specification.

End users and their requirements.

The end users will be gamers who like stealth games. They will probably be between 15-25, and enjoy sneaking around enemies and cleverly navigating a map. They want a challenge, but not something that's too hard as they would just get frustrated. They would expect;

- Solid gameplay.
    - No bugs.
    - No frame rate loss i.e run at 30 frames per second minimum.
- Responsive gameplay.
    - No delays or pauses when the user tries to do something.
    - Player's character must feel agile, not heavy and clunky.
- Competent AI.
    - No running into walls.
    - Able to navigate obstacles.
- Freedom.
    - They should be able to quit the program at any time.
- Good level design.
    - The level should present lots of possible solutions to a problem, so the player can choose.

Functional Requirements:

- Gameplay:
    - The player must sneak through a level, while staying undetected. This is critical for a stealth game, As stated in the research document.
    - The player's goal is to delete data on a server, deep in the level. It must be deep in the level to allow the user to get challenged, as stated in the user requirements.
        - The 'terminal' to access the server will be protected, so it will require the player to either hack the terminal or find a password.
        - Hacking is a minigame that involves capturing nodes in order to access an access point. The player must reach the access point before the enemy system captures all the players nodes.
        - The other way to access the server, is to find a password and username, which will involve some degree of detective work. The player must find

the admins room, log onto their computer and search a database (via binary search) for the password for maintenance staff.
    - The player can choose whether or not to kill enemies, this was identified as a need in the research document.
        - Killing enemies means that they won't be a problem later.
        - Sparing them means the player gets a larger score bonus.
    - The player can open and close doors.
    - The player can walk, run, crouch and shoot. Player mobility was identified as a need in the research document.
        - The player can shoot enemies with a pistol.
        - Crouching allows the player to sneak through vents.
    - Guards have AI. Decent AI was identified as a need by the research document.
        - They patrol around predefined waypoints when they can't detect the player.
        - When the detect the player and can see them, they shoot at the player. When they can't see the player, they move to the players last position. If they still can't see the player, then they return to patrolling. If they can still see the player they shoot.
- UI:
    - Splash screen.
    - Start menu.
    - Options menu.
        - Lets the player change graphics quality.
        - Lets the player change audio levels.
    - Mission failed screen.
        - Queries the player to try again or quit.
- Animations and models:
    - Low-poly aesthetic.
    - Player model.
        - Walk, run and crouch anims.
    - Guard model.
        - Walk, run, shoot (stationary) and an alerted anim.
    - Environment.
        - Wall model. (Static).
        - Wall edge model. (Static).
        - Door model. (No anims as movement is script driven).

        - Computer and screen. (Static).
        - Background objects. (Static).
    - Items.
        - Pistol.
            - Reload and recoil anims.

Non-Functional Requirements:

- Open source.
- Documented.
    - Installation manual.
    - Development documentation.
- Portable to PC, Mac and Linux.
    - Stretch goal: PS4 and XB1. (As Unity makes this possible).
- Good performance.
    - Run at at least 30fps on low spec machines.
    - Ability to change graphics fidelity for high-end machines.
- Possibly usability.
    - I.E controller instead of keyboard.

Scope and Boundaries.

- Scope.
    - Game will have one level to complete.
    - It will have a hacking minigame.
    - Opening and closing doors.
    - Ai.
    - First person shooter.
    - Ability to hide from the Ai.
- Boundaries.
    - To run the game at acceptable speeds the computer requires a graphics card.
    - Ai will be dumb and forgetful.
    - Only one weapon to use.
    - No music played in the game, as I can’t compose music.
    - Unable to save progress.
    - Unable to heal player.
    - Player has unlimited ammo (didn’t bother implementing ammo, would require looting if the player ran out).

Inputs and outputs.

- Input **-** Directional keys, i.e **w**, **a**, **s** and **d.**
    - Output - Moves character.
- Input - The Q key.
    - Output - Advances dialogue if there is any.
- Input - The E key.
    - Output - Interacts with whatever the player is looking at, if It can be interacted with.
- Input - Spacebar.
    - Output - Makes the player jump.
- Input - Left click on the mouse.
    - Output - Makes the player shoot.
- Input - Moving the mouse.
    - Output - rotates the camera around.

Test Plan.

Comprehensive test plan.

The table below is all of the tests to be carried out by me. I will duplicate the table and add an extra column called '**Received**' which will contain the results of what output the game gives me.

Test	**Expected**
Player Detection	
Player stands behind a wall with a guard on the other side.	Player is not detected.
Player stands behind a guard.	Player is not detected.
Player stands in front of a guard.	Player is detected after the amount of time declared on the guard.
Player enters guards vision cone.	Player is detected after the amount of time declared on the guard.
Player leaves guards vision cone.	Player is no longer detected.
A closed door is between the guard and the player.	Player is not detected.
An open door is between the guard and the player	Player is detected after the amount of time declared on the guard.
Environment	

Player is looking at an object that derives from the Interactable base class and is **within** the specified range.	'Interact' prompt appears on the players UI.
Player is looking at an object that derives from the Interactable base class and is **out of** the specified range.	'Interact' prompt does not appear on the players UI.
Player is **not** looking at an object that derives from the Interactable base class and is **within** the specified range.	'Interact' prompt does not appear on the players UI.
Player is **not** looking at an object that derives from the Interactable base class and is **out of** the specified range.	'Interact' prompt does not appear on the players UI.
Pressing 'E' while looking at an interactable and being **within** its range.	The overridden Activate() method is called on the interactable.
Pressing 'E' while looking at an interactable and being **outside** its range.	Nothing happens.
Pressing 'E' while **not** looking at an interactable and being **within** its range.	Nothing happens.
Pressing 'E' while **not** looking at an interactable and being **outside** its range.	Nothing happens.
Guard AI - State Transitions	

Guards default state.	Patrolling.
When the player **enters** the guards vision cone when guard is **patrolling**.	Change state to Suspicious.
When the player has been in the vision cone for long enough to get detected.	Change state to Attacking.
Player **exits** the vision cone when the guard is in the **suspicious** state.	Change state to Patrolling.
Player **exits** the vision cone when the guard is in the **Attacking** state.	Change state to Pursue.
Player **stays out** of the guards vision cone for a set amount of time when in the **Pursue** state.	Change state to Patrol.
Player **enters** the guards vision cone when in the **Pursue** state.	Change state to Attacking.
Guard AI - Environment Interactions	
The guard enters the trigger collision mesh of a door.	If the door is open, nothing should happen. If the door is closed then it should open.
Guard has to navigate to a point in the world.	The navMesh generates a path, and the guards motor makes it follow it.
Guard has to navigate to an **invalid** point in the world (i.e. its not reachable).	Guard reverts to the patrol state.

Hack Puzzle	
CreateGrid() called.	Nodes should appear as defined in the script. LineRenders should appear, visually connecting them together as defined in the script.
Node instantiated.	Node intro animation plays.
Node type changed.	The current gameObject is deleted and a new one of the correct type is instantiated.
Player uses the W, A, S and D keys to move the cursor.	Cursor moves around the grid, without leaving its boundaries.
Player presses Esc.	Hack puzzle closes.
Player presses Spacebar and the cursor is on top of a **Controlled** node.	Error beep sounds.
Player presses Spacebar and the cursor is on top of a **Uncontrolled** node.	The CaptureNode(Node a) Coroutine is called and the cursor cannot move until the node has been captured.
Player presses Spacebar and the cursor is **on top** of a **Firewall** node.	Error beep sounds.
Player presses Spacebar and the cursor is **on top** of the **Access Point** node.	The player has won. EndHack(bool _successful) is called with _successful being true.
Player presses Spacebar and the cursor is **on top** of a	Nothing happens.

Null node. (This has no visual representation).	
All of the players nodes are captured by the firewall.	The player has lost. EndHack(bool _successful) is called with _successful being false.

Input validation testing.

These are a list of tests that will be used against the 'Data server' that takes in a string of letters and numbers in the form 123-4AB.

Test	**Expected**
123-4AB	ID accepted.
ABC-DEF	ID rejected.
123-ABC	ID rejected.
ABC-123	ID rejected.
123-4£&	ID accepted.
$%&-4AB	ID rejected.

End user testing.

These are a list of questions the end user will be asked after they play the game.

- Was it enjoyable?
- Were there any minor bugs that ruined your experience? If so please list them.
- Were there any major bugs that crashed the game? If so please list them.
- Did you feel challenged? Did you feel *too* challenged?
- Were you engaged?
- Did you feel that the game's stealth mechanics could stand up to its other competitors mechanics?

Design.

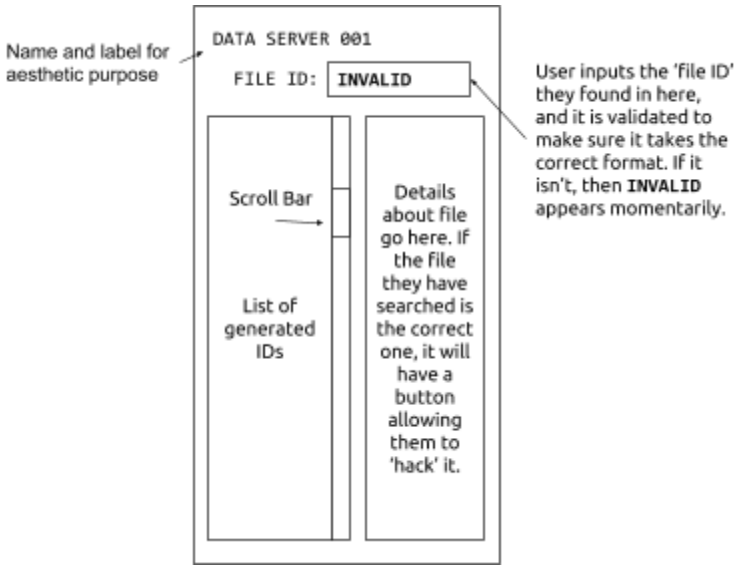
Design methodology.

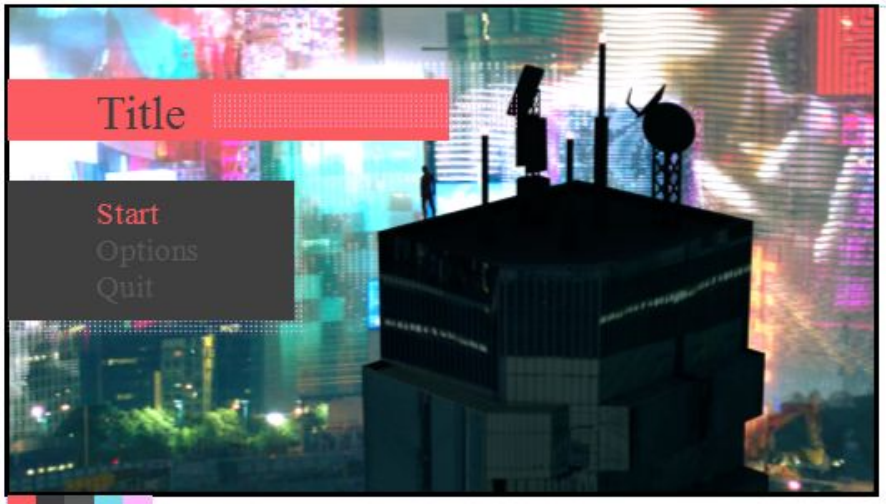
I have chosen the **agile** methodology, based on the iterative cycle of design, developing and testing. It's much faster and more flexible than the waterfall method, and as I'm the only one working on it, there would be no lack of communication. It also allows me to test bits of the game as soon as they are working, instead of programming everything at once and then testing, which lets me refine and fix code and architecture problems as soon as they are implemented, instead of a month down the line. Overall this saves time and effort. If I had a paid license for Unity or a modeling program like Maya, I would save money as well, but everything I am using is free so the financial aspect doesn't matter.

UI design.

Data server UI.

The data server interface pops up when the player (user) interacts with the data server. They can see all the file ID's available, but there will be so many that guessing which one to pull up would be boring. When they input an invalid ID, the input field will be cleared and the text 'invalid' will fade in and out beside it. When they get the correct ID it will pull up details about the file (something like its encrypted, that means they have to hack it) and the user can start the hack puzzle minigame. The design should be futuristic, as the end users would want that sort of aesthetic. The input field would expect a string like 123-4AB, where numbers must be numbers and letters must be letters, e.g. 045-5WG. The wireframe for the interface is below;





Keep in mind that the background is a render, and would have to be reconstructed inside Unity. This makes it difficult to get lighting correct, but also means I can animate everything to make it more interesting.

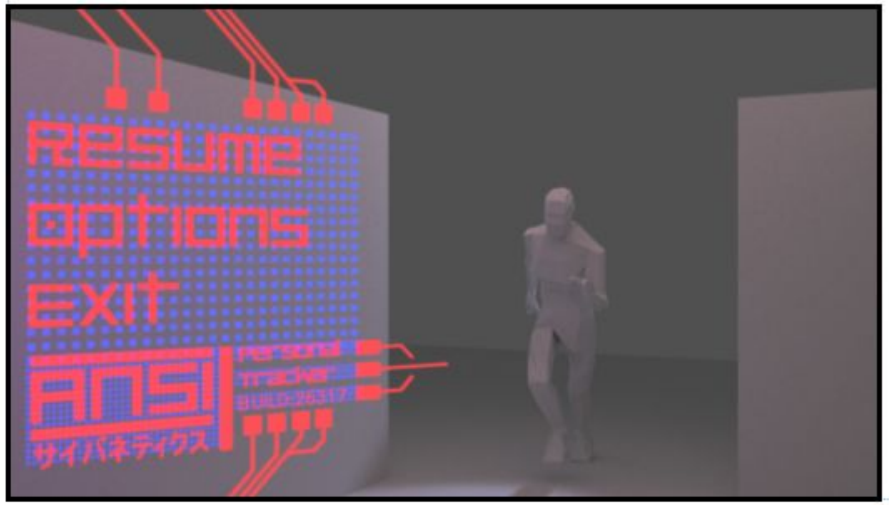
ANSI サイバネティクス
START
OPTIONS
QUIT

A good UI tells you what **you need to know, and then gets out of the way**.

1. Does this interface tell me what I need to know *right now*?
2. Is it easy to find the information I'm looking for, or do I have to look around for it? (Are the menus nested so deep that they hide information from the player?)
3. Can I use this interface without having to read instructions elsewhere?
4. Are the things I can do on this screen obvious?
5. Do I ever need to wait for the interface to load or play an animation?

Are there any tedious or repetitive tasks that I can shorten (with a shortcut key, for example) or remove entirely?

Note:
The pause UI should look like a sologram inside the players eyes, and tilt slightly outwards.



Program/Data structure design.

Data server.

The data server is a game object the player(user) interacts with. They use the 'file ID' (a random string of letters and numbers) that they find earlier in the level to access data and win the game. The script that sits on the data server is the script that carries out a bubble sort and has input validation.

The 'file ID' that the player inputs is validated to prevent a crash. The string should take the form 123-4AB, where numbers must be numbers and characters must be characters. The dash in the middle is also required.

C style pseudocode for validation (https://en.wikipedia.org/wiki/Pseudocode).

```
If length of input does not equal 7 characters {
        Return
}

Turn the input string into a character array

Set firstHalfTrue to if input[0] is an integer and if input[1] is an integer and if input[2] is an integer;
Set dashValid to if input[3] is '-';
Set secondHalfTrue to if input[4] is an integer and if input[5] is a character and if input[6] is a character;

If firstHalfTrue and dashTrue and secondHalfTrue {
        The search key is valid;
}
```

C style pseudocode for sorting.

```
for integer outerLoop = length of generated IDs - 2; outerLoop > 0; outerLoop--) {
        for integer i = 0; i < outerLoop; i++) {
                if generated IDs [i] precedes generated IDs [i+1] {
                        Swap position i and i+1;
                }
        }
}
```

Hack puzzle design.

When they player attempts to access the file, or unlock a door, they will be greeted with the hacking puzzle. The objective of the hack puzzle is to capture the access point. This allows the play to gain access to whatever the hack puzzle is attached to, like a locked door or server. The grid is a 5x5 2D array that holds information on the nodes. There are 4 types of node; Controlled (by the player), Uncontrolled, Firewall and Out Port.

- Controlled nodes are the nodes that the player has captured.
- Uncontrolled nodes are nodes that have not been captured by the firewall or player.
- Firewall nodes are nodes controlled by the firewall. The firewall is an AI whose purpose is to capture all of the players nodes, in which case the player loses. Initially, the AI will be primitive, capturing every node it can. If I have time I'll design and implement a state machine similar to the guards.
- The Out Port node is the node that the player captures to win and unlock whatever the hack puzzle is attached to.

Nodes can paired to other nodes making a connection, represent to the player by a line, and represented in code by a NodePair structure. The player can only capture an uncontrolled node that has a connection to a controlled node, and the firewall can only capture nodes that are connected to any of its firewall nodes. Connections and Node types are defined in the script itself, however in later version a 'cartridge' system could be implemented, allowing different setups to be plugged in via the Unity Editor.

<Put in a picture>

I've considered many mechanics for capturing nodes. I have chosen two and will decide on which one to use after I see how fun or challenging.

- Player has to complete some kind of puzzle or a quick time event.
    - This means how fast the player progresses is up to the player.
    - Potentially frustrating.
- Player has to wait a certain time to capture a node.
    - This means that the speed that the player progresses is fixed.

- Easier.
- The AI will use this method, as a computer matching numbers would be unfairly fast.

Data structure.

Node types are defined as an enum.

```
enum NodeTypes {nullNode, controlled, ... ,
AccessPoint};
```

Nodes will have their own structure that hold their type, location and the associated game object.

```
struct Node
{
	public NodeTypes type;
	public Vector3 location;
	public GameObject gameObj;
	public int x;
	public int y;
}
```

The grid will be a 5x5 2D array of type Node.

```
Node[,] grid = new
Node[5,5];
```

The connects between nodes will be a structure that holds two nodes and the line renderer script associated. It has to be a class because the pairs will be compared like objects.

```
class Pair
{
      public Node a;
      public Node b;
      public LineRenderer line;
}
```

The pairs will be tracked and stored in a list.

```
private List<Pair> NodePairs = new
List<Pair>();
```

I will implement 5 node functions;

```
void ConnectNodes(Node a, Node b) { ...
}
```

- This will create a Pair between the two nodes passed in.

```
bool IsConnected(Node a, Node b) { ...
}
```

- This will return true if a pair exists between the two nodes.

```
List<Node> GetConnectedNodes(Node node, NodeTypes type) { ...
}
```

- This returns a list of nodes that are connected to the node passed in, that are of the type passed in. If null is passed in as the type then all nodes connected are returned regardless of their type.

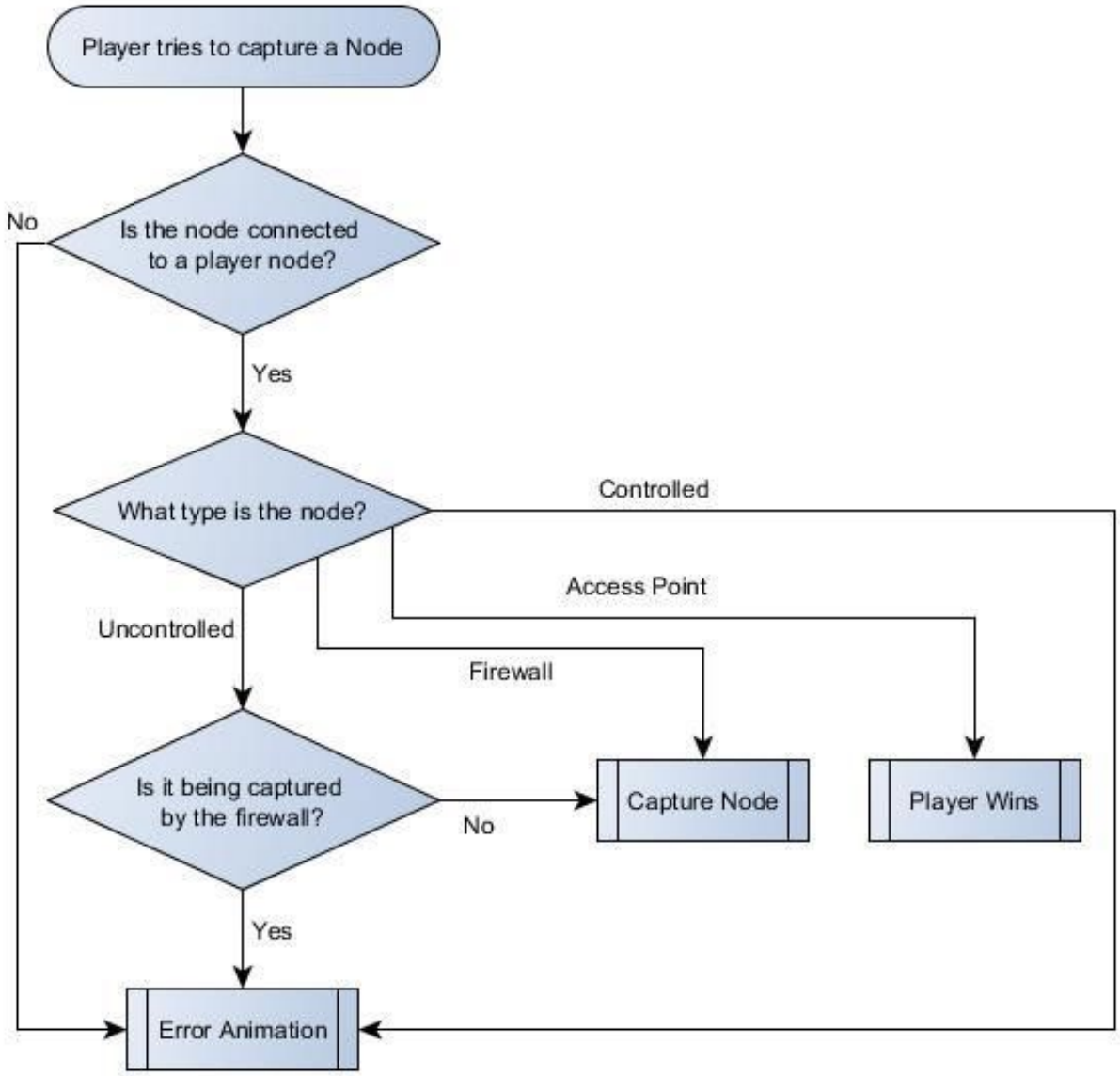
```
GameObject CreateNodeGameObj(NodeTypes _type, Vector3 _pos) { ...
}
```

- This creates the node in-game so the player can see it. It returns the gameobject of the node specified, and it will be at _pos in worldspace.

```
void ChangeNodeType(ref Node _a, NodeTypes _type) { ...
}
```

- This changes the type of the node passed in to the type passed in. A method is needed for the deceptively hard command because it's gameobject also needs to be updated, and is also the reason that `CreateNodeGameObj()` exists.

Game flow is as follows;



AI Design for the hack puzzle.

To start with, the AI will be primitive, it's only goal is to capture connected nodes at random. This is because navigating the connected nodes would be too difficult as the representation of the grid in the script is too abstract (as its connections are just lists). If I were to create an AI that thinks about what move to make, I would have to research further into the topic and I can imagine that it would be very resource heavy, unless I refactor they way the grid is represented. So the first iteration of the AI will be a coroutine that captures a random node connected to a firewall node, then waits for a tweakable amount of time, and capture another node.

Initialization for the hack puzzle.

The hack puzzle is launched by a script (this could be anything like a keypad or a button) calling a function called CreateGrid(). This starts the hackpuzzle. The player motor should then be paralysed (input doesn't move it), the cursor is freed and the main camera should switch to a secondary camera that doesn't move and is in a good position to play the puzzle from (It should also be set not to render the player incase the model is in the way).

Ending the minigame for the hack puzzle.

If the player captures the AccessPoint, then the hackpuzzle calls a EndHack(bool _success) method on whatever started the minigame, with _success being true. If all player nodes get captured then EndHack() is called with _success being false. The whatever started the minigame can deal with the outcome however it wants (A door could unlock upon success, or an alarm could go off upon failure). The player can also press escape at any time, and the hack will be 'aborted' i.e the hackpuzzle will close down but EndHack won't be called.

Dialogue Design.

The original dialogue system was designed by Brackeys - https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA

I extended it so it fitted my needs using scriptable objects.

The dialogue manager script is the object that actually controls what dialogue is being displayed on screen. It has two main functions:

```
public void LoadDialogue(Dialogue _newDialogue) { ...
}
```

Which takes in the dialogue object to be displayed and loads all of its dialogue into a queue, and:

```
public void DisplayNextSentence() { ...
}
```

Which (funnily enough) displays the next sentence from the queue.

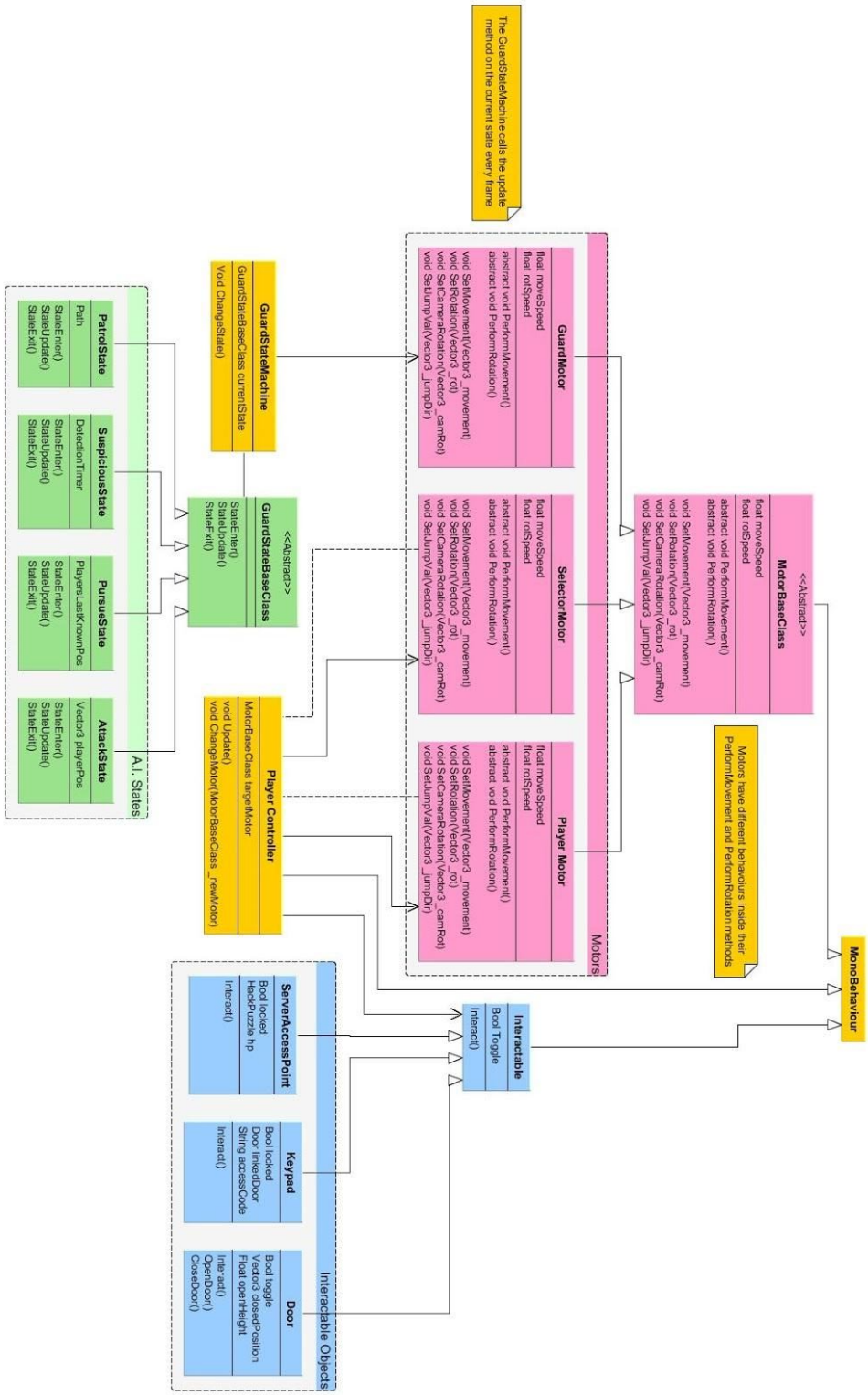
A Dialogue object is an object with custom properties that I've created:

```
[CreateAssetMenu(fileName = "newDialogue", menuName =
"Dialogue")]
[System.Serializable]
public class Dialogue : ScriptableObject{
      public string npcName;
      [TextArea(3, 10)]
      public string[] sentences;
}
```

This allows me to create a bunch of objects of this type with their own details for npcName and sentences[ ].

Load dialogue can be called by anything. From a button press to a animation finishing.

Class diagram for everything.



Implementation.

Final testing of solution.

- Player sneaking through a level.
    - Enemy detects player when the player is in its vision cone for the time specified in its behaviour.
        - This will be tested by having the enemy's vision cone visible in play mode via gizmos and then timing how long it takes for the enemy to change state.
    - Player can hide behind objects.
        - This will be tested by having a static enemy looking at a wall, and the player positioned behind it. A debug statement will be set up to see if the enemy can see the player, which it shouldn't.
    - Player can open doors.
        - This will be tested by having the player open and close a door as much as they want. The door should stop if something is in the way.
    - Player can open vents.
        - Tested the same as doors.
- Player's goal is to ~~delete~~ 22/02/18 retrieve data on a server.
    - The player can access the ~~terminal~~ 22/02/18 server.
        - This will be tested by seeing if when the player presses 'E' on the keyboard and the player is looking at an interactable, ~~the 'interact' method~~ 22/02/18 the 'interact' method inherited in from its base class is called and the player should be immobile and the ~~camera should reposition to a certain point and look and the terminal.~~ 22/02/18 the active camera will switch to a secondary static camera position before the game started.
        - The player will need to exit the ~~terminal~~ 22/02/18 server at one point, so pressing escape should result in the player returning to their body. And be able to move around.
        - While the player is accessing the ~~terminal~~ 22/02/18 server, the hacking minigame puzzle should run. This will be tested by observation.
    - The player must complete a hacking minigame to access the information.

        - The minigame should ~~read in a bitmap~~ 23/02/18 get node positions and connections as they are described in code and create a 'board' from it. This will be tested by observation.
        - The player can control a cursor that snaps to the same grid as the nodes. This will be tested by having the cursor read out its positions to the console.
        - Pressing ~~'E'~~ 23/02/18 space while in the minigame should turn an uncontrolled node into a controlled node, if it is connected to a controlled node. This will be tested by observation and console output.
        - The 'firewall' should capture controlled 23/02/18 and uncontrolled nodes provided that they are connected to a 'firewall' node.
        - If all the players nodes are captured by the firewall, the player is rejected and returns to their body.
- UI.
    - Main menu.
        - The start button should load the game scene.
        - The options button should load the options scene.
        - The quit button will exit the program.
    - Options menu.
        - Entering a volume value larger than 100 and less than 0 should be prevented.
- Animations.
    - A guard walking forward should play the walk anim.
    - When a guard is stationary it should play the idle anim.
    - When a gun is out of ammo and the user attempts to fire, they will have to wait for the reload anim to finish.

Results will be recorded in a table, and components will be tested during their development. The game will regularly be distributed to beta testers and their reports will be collected. The finished game will be distributed to the end users and they will voice their opinion.

23/02/18

Test	**Expected**	**Received**

Player Detection		
Player stands behind a wall with a guard on the other side.	Player is not detected.	Player is not detected. 23/3/18.
Player stands behind a guard.	Player is not detected.	Player is not detected. 23/3/18.
Player stands in front of a guard.	Player is detected after the amount of time declared on the guard.	Player is detected after the amount of time declared on the guard, specifically, 1 second. 23/3/18.
Player enters guards vision cone.	Player is detected after the amount of time declared on the guard.	Player is detected after the amount of time declared on the guard. 23/3/18.
Player leaves guards vision cone.	Player is no longer detected.	Player is no longer detected. 23/3/18.
A closed door is between the guard and the player.	Player is not detected.	Player is not detected. 23/3/18.
An open door is between the guard and the player	Player is detected after the amount of time declared on the guard.	Player is not detected. 23/3/18. Player is detected after the amount of time declared on the guard. 23/3/18. The collider on the door was blocking the guards raycast.
	Systems working correctly?	Yes. 23/3/18.
Environment		
Player is looking at an object that derives from the Interactable base class and is **within** the specified range.	'Interact' prompt appears on the players UI.	'Interact' prompt appears on the players UI. 24/3/18.

Player is looking at an object that derives from the Interactable base class and is **out of** the specified range.	'Interact' prompt does not appear on the players UI.	'Interact' prompt does not appear on the players UI. 24/3/18.
Player is **not** looking at an object that derives from the Interactable base class and is **within** the specified range.	'Interact' prompt does not appear on the players UI.	'Interact' prompt does not appear on the players UI. 24/3/18.
Player is **not** looking at an object that derives from the Interactable base class and is **out of** the specified range.	'Interact' prompt does not appear on the players UI.	'Interact' prompt does not appear on the players UI. 24/3/18.
Pressing 'E' while looking at an interactable and being **within** its range.	The overridden Activate() method is called on the interactable.	The overridden Activate() method is called on the interactable. 24/3/18.
Pressing 'E' while looking at an interactable and being **outside** its range.	Nothing happens.	Nothing happens. 24/3/18.
Pressing 'E' while **not** looking at an interactable and being **within** its range.	Nothing happens.	Nothing happens. 24/3/18.
Pressing 'E' while **not** looking at an interactable and being **outside** its range.	Nothing happens.	Nothing happens. 24/3/18.
	Systems working correctly?	Yes. 24/3/18.
Guard AI - State Transitions		
Guards default state.	Patrolling.	
When the player **enters** the guards vision cone when guard is **patrolling**.	Change state to Suspicious.	Change state to Suspicious. 26/3/18.

When the player has been in the vision cone for long enough to get detected.	Change state to Attacking.	Change state to Attacking. 26/3/18.
Player **exits** the vision cone when the guard is in the **suspicious** state.	Change state to Patrolling.	Change state to Patrolling. 26/3/18.
Player **exits** the vision cone when the guard is in the **Attacking** state.	Change state to Pursue.	Change state to Pursue. 26/3/18.
Player **stays out** of the guards vision cone for a set amount of time when in the **Pursue** state.	Change state to Patrol.	Change state to Patrol. 26/3/18.
Player **enters** the guards vision cone when in the **Pursue** state.	Change state to Attacking.	Change state to Attacking. 26/3/18.
	Systems working correctly?	Yes. 26/3/18.
Guard AI - Environment Interactions		
The guard enters the trigger collision mesh of a door.	If the door is open, nothing should happen. If the door is closed then it should open.	Door open - nothing happens. 28/3/18. Door closed - AI opens door. 28/3/18.
Guard has to navigate to a point in the world.	The navMesh generates a path, and the guards motor makes it follow it.	NavMesh generated when guard enters pursue state, and commands motor to follow. 28/3/18.
Guard has to navigate to an **invalid** point in the world (i.e. its not reachable).	Guard reverts to the patrol state.	Game crash. 28/3/18.
	Systems working correctly?	

Hack Puzzle		
CreateGrid() called.	Nodes should appear as defined in the script. LineRenders should appear, visually connecting them together as defined in the script.	Nodes appear in the wrong orientation. 19/2/18. Nodes appear in the right orientation, but there are no lines connecting them. 20/2/18. Nodes appear as defined with lines connecting them. 20/2/18.
Node instantiated.	Node intro animation plays.	Anim plays. 19/2/18.
Node type changed.	The current gameObject is deleted and a new one of the correct type is instantiated.	gameObject is deleted, but it doesn't update properly? 19/2/18.
Player uses the W, A, S and D keys to move the cursor.	Cursor moves around the grid, without leaving its boundaries.	
Player presses Esc.	Hack puzzle closes.	
Player presses Spacebar and the cursor is on top of a **Controlled** node.	Error beep sounds.	
Player presses Spacebar and the cursor is on top of a **Uncontrolled** node.	The CaptureNode(Node a) Coroutine is called and the cursor cannot move until the node has been captured.	
Player presses Spacebar and the cursor is **on top** of a **Firewall** node.	Error beep sounds.	

Player presses Spacebar and the cursor is **on top** of the **Access Point** node.	The player has won. EndHack(bool _successful) is called with _successful being true.	
Player presses Spacebar and the cursor is **on top** of a **Null** node. (This has no visual representation).	Nothing happens.	
All of the players nodes are captured by the firewall.	The player has lost. EndHack(bool _successful) is called with _successful being false.	

Evaluation.

Evaluation of the solution.

My project's objectives and requirements changed a lot over the development process. The actual data structures changed (Some AI states were meant to be recursive) and the actual game content changed as well (only one weapon to choose from, and no art assets). The data structures changed because as I was creating the game, my main goal was to be very disciplined over the architecture of the game. Some structures just didn't make sense / were to awkward to use.

Change	**Justification**
Guards use their own motor instead of one that derives from the motor base class.	This is because the motor base class is more geared up for player input, instead of A.I input. The motor base class implemented lots of functions like 'Move()' and 'Jump()' which took in vectors, that would be complex for an A.I to generate and get feedback from. So a much cleaner custom motor was made that had functions like 'MoveTo()' which took in a *position* instead of a direction, and would return false if the motor was already at that position. This allowed the A.I to have more control.
Patrol state no longer uses recursion for patrolling between points.	This is because it made little sense with the StateUpdate() method, which is called every frame. Plus recursion has some overhead, and with it calling itself ***a lot*** (every frame that passes, the method would call itself. That could be up to 50,000 times) which would eventually cause the program to crash or run out of memory.
Hack puzzle no longer reads in a bitmap of node positions, instead it's defined by several method calls at the start.	This was because that writing an algorithm to decipher nodes and their connections would be way to time consuming. The only reason this was in in the first place was to be the programming feature that interfaced with stored data. As this it's no longer required and would be to difficult to implement, it was scrapped.

A dialogue system was added.	This was because the player needed some informing in-game on their objective. It also uses a queue programming structure, which means I can bump off one of the programming features which don't really fit and make the development process easier.

How closely does it meet its requirements specification?

It only loosely matches the original requirements spec. However, it matches the new one perfectly. The requirements spec was changed over the course of the development process to accurately reflect what was possible in the timescale and what I determined as irrelevant.

Evaluation on the final testing carried out.

The final testing was comprehensive and in depth only in certain areas. Some tests were way to specific in areas and to sparse in others. Also, a lot of testing was done during development that went unrecorded as it was on the fly. Overall, my testing performance was poor. If I had went with the waterfall method, the project would likely be a total car crash, instead the agile methodology allowed me to fix bugs very quickly, as well as Unitys error tracebacks and community help.

Evaluation on the end-user testing carried out.

There were only 7 questions in the questionnaire, as I found it difficult to come up with relevant questions for a video game. However I felt that the questions raised covered enough content to be considered satisfactory. The end users reported little to no bugs, and the bugs that were there weren't game breaking (UI scaling issues, which had went under my radar because I use a 16:9 screen, but the end user's screens were 4:3, which cause some UI elements to become unaligned or invisible). 6 end users were tested, and they all enjoyed the game, which was the games main purpose. They felt it was solid, and that it ran at a stable fps of around 40.

Further developments.

Further developments would probably be 3D assets to decorate the level. These were in the original requirements specification but were cut out because of time restraints. This would make the game much more 'immersive' and pretty to look at. The second further development would be some of the game mechanics that were originally proposed, like moving objects to avoid detection and guards being able to hear the players movements. Each system would be a project in itself, and require months to properly design and develop.

Evaluation of critical points.

Critical Point	**Evaluation**
A.I implemented and working.	This was a major confidence boost, and a lot of stress was taken off (because if I couldn't manage to program A.I, it would be a pretty boeing game and leave a lot of the end user requirements unfulfilled). The A.I design itself follows a structure that was found during the research I did on A.I, however I had to modify it as it used pointers, which C# doesn't properly support yet. Instead of using a pointer to an A.I state, the gameobject get the script added as a component. Looking back however, I reckon that I could've used scriptable objects instead and used them like 'cartridges' that can be plugged in to the controller script.
Player controller and motor working.	I could've used the Unity's default character controller, but thats boring and handles a bit funny, so I wrote my own, that fits in with my architecture. It's ok, and it makes *some* sense, but implementing new things is a hassle, for instance the pause menu. That was a mess. My discipline is that the player controller handles all input from the user, and the player motor handles physical interactions. However I'm sure there's a script somewhere that handles input on it's own, which goes against the discipline.
Interaction system implemented.	I'm not going to lie, I think this systems kinda cool. As long as an object has a script attached to it that inherits from the interaction base class, it can be interacted with. This makes setting up and interactable really easy, as well as very customisable. I'll definitely be exporting these scripts as a UnityPackage so I can put them in future games.

Overall conclusion.

Overall, i'm happy with how to projects turned out. It has responsive and solid gameplay that meets the end-users requirements, and they seem happy enough with it.

Fitness for purpose, I.e is it fun?

The end-users say that it's fun, so it's purpose is realised. The main task of my project was to create a stealth game for a player to enjoy. The game I ended up creating *is* a stealth game that fully meets its amended purpose, and the players do enjoy it.

UI.

The UI in the game is far from the original design. However, it still has the same buttons and links and the original, it's just that the art style changed. The pause screen allows player to return to the main menu or return to desktop. The options menu wasn't implemented as I ran out of time. The UI is straightforward as there are no sub menus. It's easy to find the information the user is looking for.

Robustness.

Overall, the game is robust. There will not be any 'missing file' because a directory is wrong, as the game is 'built' into a package that works on its own. However, if the user deletes the file the accompanies the executable, unsurprisingly, the game won't work. But that is the same with every program. The program is very unlikely to encounter uncorrectable errors during runtime.

Reliability.

Overall, and after a couple of 'patches', the game is mostly reliable. The biggest bug that the end users encountered was that if they lost the hack puzzle, the UI would freak out. The 2nd biggest bug was that if the player was engaged on something (like accessing the computer), the pause menu would act all janky. I quickly patched this issues and they haven't been reported since. The 3rd bug was that if a guard detects you and follows you, it doesn't pathfind back to its waypoint, it just points in its direction and walks. So if there is a wall between it and its waypoint, then it runs into it. However I was aware of this issue when I finished the AI, I just never got round to implementing pathfinding for it. I would patch it as a further development.

Portability.

Unity allows me to build to almost any platform, but the two current builds that exist are Mac OS 64bit and Windows 64bit. I can port to more, like a PS4, but that requires a license, which is expensive.

Efficiency.

The program is efficient enough to run properly, however there are a lot of function calls at the start, where every script looks through everything looking for the components it uses. This doesn't cause any delay on *my* machine, but it might cause lower end machines to choke up. This could be avoided by making scripts like GameManager and HUDmanager singletons or hardcoding the components into the scripts.

Maintainability.

I've worked really hard to make sure the architecture of the code base was well organised and made sense. This means that adding a feature is easy, I just have to be disciplined when adding stuff and make sure it's as tidy and clean as everything else, otherwise it would just become a huge mess. All the classes are in individual scripts, so even global variables aren't that bad. All variables have meaningful names, with being too long. I also have a practice where I put an underscore before a variable if it exists only in a function or subprogram, this allows the code to be more readable as you know that certain variables have a certain scope. For instance;

```
string globalVar;

void foo()
{
    string _localVar = "bar";
}
```

I believe i've taken the most 'proper' solutions to all the problems the project had. I.e solutions that were as simple as possible while still doing the job properly. No waffle, but no corners cut either.