Logan Broomhead-Thompson, GEP Report

# Core Concept

A top-down stealth game, that gets the player in touch with their inner B-movie hacker.

There are two screens and gameplay flavours. One for hacking. One for stealth.

Stealth:

A top-down camera perspective. Security cameras and guards pose as threats for the player. Entering their sight line is a game over.

Guards follow a patrol route. The player should move slower than the guards to force them to learn the patrols and carefully operate around them.

Security cameras are rotate on walls. They can be hacked by clicking on them. Once hacked, a camera can be turned on or off. There should be parts of the level that require you to hack a camera.

Doors will block the players progress, forcing them to hack it. (Alternatively, put a security camera in a corridor that can’t be snuck around.)

Whilst hacking, the player cannot move. This forces them to find a good hiding spot away from guards. An ideal level layout should encourage the player to hack a camera so they have a spot to hide from guards, whilst getting close enough to hack a door.

Hacking:

Hacking should be an equally vital part of the game. It should encourage the player to type on their keyboard as quickly as possible. The speed of their typing correlates to the speed of the hacking.

An image of a keyboard appears on the screen and random keys flash. Typing a flashing key gives hack progress. The timing window for the flash should be short enough that relying on reaction time alone isn’t enough to hit all the keys. The player will want to type as quickly as possible in the hopes of getting lucky. The flash should be long enough that some keys can be taken with reaction speed if the players fingers are near the corresponding key. (This gives the system a bit of skill expression)

To make the system more interesting. Pop ups will occasionally appear and slow your progress. The player must click on them with their mouse to get rid of them. This will impact their ability to hit the keys. The player will have to decide when to grab their mouse and delete the pop ups. This will give an ebb and flow to the hacking system as the player moves between typing (reaction speed challenge) and clicking (mouse movement accuracy challenge.)

The pop ups should further sell the idea of being a B-Movie hacker by using language like “Mainframe Firewall”, and “Router Point Data.”

# Mechanics and Code

## Two Screens

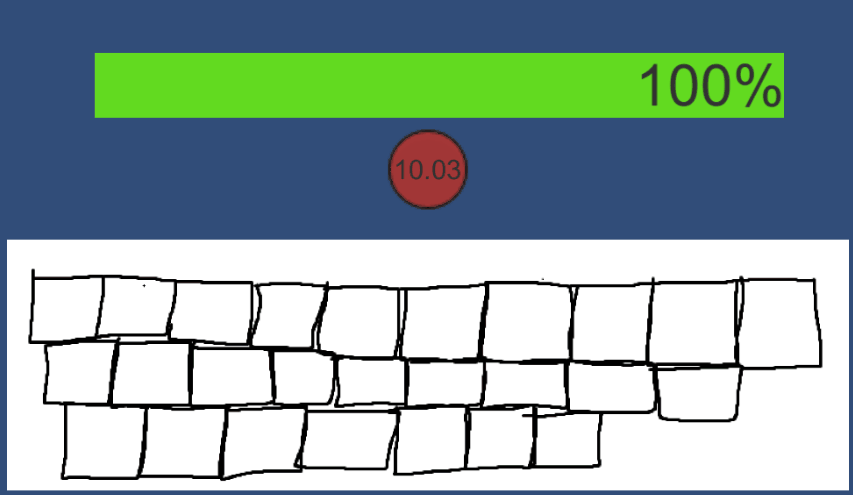
My gameplay is split between two main screens.

Graphical user interface, application

Description automatically generated

Stealth gameplay is recorded in a separate scene and sent to a render texture which only captures layers from that scene. This way the screen is limited to only a small portion of the display. I could have used a mask but I found it tedious to change the settings on every sprite.

## Hacking Gameplay



There is a lack of visual feedback to this system. It’s unclear for an observer what keys the player is clicking. Good feedback would also make the game feel better to play.

I’m aware that the timer here could use a lot of improvements. It gets covered by the pop ups, and it should be clearer how long the player has left at a glance. Showing the timer with just a number isn’t good enough. Plus its not obvious that that red circle is even a timer until it starts ticking down.

## Hackable Manager

**HM\_HackingManager, HM\_KeyboardKey, HM\_PopUpScript, and Difficulty**

|  |  |
| --- | --- |
| HM\_Hacking Manager |  |
| Progress | Tracks player progress. Function to change progress which also checks if its >= 100 |
| Begin hacking | Starts a hacking minigame, and remembers which object called it. Once the hack is completed, this object will become interactable |
| Complete Hack | When the progress meter is full, the object that started the hack will become interactable for the player |
| Enable Keys | Has a list of keyboard keys, at set intervals it will turn these keys on. Enabling their scripts |

|  |  |
| --- | --- |
| Keyboard Key |  |
| Key Value | What character on the keyboard is this |
| Pressed | When the keys corresponding button has been pressed, it will send a message to the hacking manager to gain a small amount of progress, and then the key will de-activate itself. |
| De-activate | The key should only be enabled for a small amount of time. A short enough time that clicking the key based on reaction time alone won’t be enough (the goal is to have the player spamming the keyboard at all times) |

|  |  |
| --- | --- |
| Pop ups |  |
|  | Distracting pop ups appear and reduce the players hacking progress |
| Destroy | Destroys itself on click, it tells the hacking manager that this happened as the hacking manager will keep track of how many pop ups there are. |

When the player clicks on a hackable object in the world, it’ll start a hacking minigame. It was my intention that every hackable object would have a glow around it so signal to the player that they can hack the object, but I didn’t get that far.

This diagram is an early plan of the hacking manager. When programming it changed slightly. For instance I planned to have the pop up scripts erase the players progress, but I found it easier for the hacking manager to simply remember how many pop ups are active, and erase progress in one go.

Diagram

Description automatically generated

A hacking manager is central to the hacking minigame. Its responsible for tracking the players progress, starting and ending the hack, as well as communicating with the other scripts related to hacking. The hacking manager is uniquely good at communication because it’s a singleton which allows objects to easily hold a reference to it.

Graphical user interface, application

Description automatically generated

The hack starts when the player clicks on a hackable object in the scene. Each hackable object holds data inside a scriptable object about its difficulty. This includes the amount of progress gained per key press, the time limit, the spawn rate of pop ups, and the amount of progress lost per second per active pop up. These scriptable objects allow me to make different difficulty settings and adjust them with ease. Originally, I was using an enumerator for easy, medium and hard difficulties. This method was far less flexible and relied on frequent if statements. A use case of the scriptable object might be that I want a separate difficulty for different security cameras depending on what part of the level they’re in. I may want a separate difficulty for doors too (maybe doors have a longer time limit but take longer to complete.)

The keyboard key represents one key on the keyboard. At slightly irregular intervals, the hacking manager will tell a random key to come alive. If the player presses that key within a specific time frame, it’ll send a message back to the hacking manager to gain progress.

Pop ups are spawned in at slightly irregular intervals by the hacking manager. The hacking manager keeps track of how many pop ups are active on the screen at once and demerits the players hack progress based on that number. The pop ups will delete themselves when they’re clicked on, and tell the hacking manager that they’ve disappeared.

I later added in animations that I wanted to play when the hack starts or stops. I used co-routines to wait for the animations to be over - it just waits 1 second, the length of the animation - because I couldn’t figure out how to use event nodes in the animator. If I ever changed the length of the animation, this could cause a problem.

## Hackable Objects

**HM\_HackableObject, IInteractables, O\_SecurityCamera, and PC\_CameraController**

|  |  |
| --- | --- |
| Hackable Object |  |
| Interact | player clicks on an object that can be hacked to toggle its functionality. (Open / Close door) If the object hasn’t been hacked yet, it will send a message to the hacking manager to start the hacking minigame |

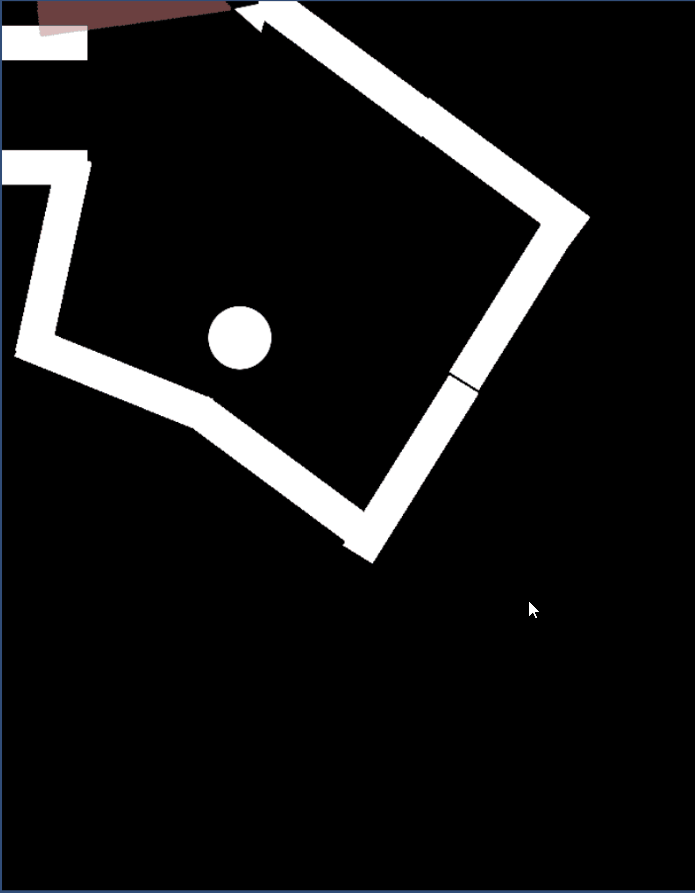
Any object in the scene that can be hacked requires three things. A collider for the player to click, the hackable object script, and a object script that describes its functionality. (E.g. Security Camera, Door)

The object script must use the interface “IInteractable.” When clicked, it’ll trigger its “Interact” function. The hackable object class remembers if the object has been hacked or not before. Every time the interact function is triggered, it’ll check with the hackable object. If it has been hacked before (the player has control of it) then the objects functionality will happen. (The security camera will turn on/off. The door will open/close.) However, if the object hasn’t been hacked yet, the hackable object class will trigger the hacking minigame. The hackable object class holds the data about the difficulty of the hack.

I chose to split the hackable objects between two scripts because I knew that every hackable object would need the same functionality: A Boolean to remember if its been hacked before, and the ability to start the hacking minigame. Each object is unique in what it does when interacted with though, so a unique script is needed for each one.

## Top Down Camera

**PC\_PlayerCameraController**



As you can see from the above gif, the player camera moves with both the players body, and the mouse.

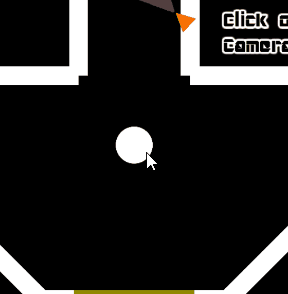
The stealth game takes place in a separate scene. This scene is captured onto a render texture and displayed in the main scene.

Having the camera follow the player was simple, but applying the offset based on the mouse location was challenging. Since the stealth scene only displays on a specific section of the screen, I needed to find where the mouse was on the render texture, rather than what percentage along the screen it was.

Later on, when I was implementing the functionality of clicking interactable objects, I realised that you couldn’t click them because they were hidden inside a different scene and displayed by the render texture. Thankfully, because I’d already figured out where the players mouse was in relation to the render texture, I was able to convert it into world space in the other scene and calculate if the players mouse clicked on any objects in the scene. I do use a Physics2D OverlapCircleAll function for this, so the code might get messy if two interactable objects were placed on top of each other. There is no reason that should happen during normal gameplay, so it didn’t feel like an issue worth investigating.

## Detection Field

**SM\_DetectionField**



Both guards and security cameras have a detection field. This shoots out raycasts in a cone looking for the player. The raycasts are only enabled if the player is close to the detection fields origin point so processing power isn’t wasted.

If the player is found, there is a short forgiveness window. After which it tells the game manager that its game over.

The detection field visualiser is incorrect. It’s just a sprite. I used raycasts because they can be stopped by walls, but the visualiser goes over them. I’m not sure how to fix this. The visualiser is a tiny bit larger than the actual radius and distance of the raycasts, so the player doesn’t feel like hitboxes are working against them.

To solve the visualiser, I could draw a line from the raycasts origin, and the place it hits. Since the raycasts aren’t cast when the player isn’t close, I’d need to increase the range at which the raycasts are shot so they always happen when the player camera could possibly be looking at the detection field.

## Guard Patrol

**SM\_GuardPatrol**

Shape

Description automatically generated

The guards have an array of patrol locations they move between. Once they reach the end, they head back towards the start. I would have liked to also implement a way for the guards to move in a loop (instead of 1>2>3>2>1, it could be 1>2>3>1…) as there are points in the game where I’d like the guards to move in a circle

The guard script uses a switch statement to change behaviours between rotating and moving. I learn about switch statements this semester and I’ve found them very useful.

I would’ve liked to add in an investigation state, where the guard moves to investigate a noise, they hear. This could be used by the player to manipulate the guard’s path, but it would also create more of a challenge as the player has to consider their footsteps.

A chase state would have been nice too. Instead of ending the game when the player is detected by a camera or guard. Nearby guards could be called to chase the player. Causing a sudden spike of drama. This would cause the gameplay to weave between two forms of interaction, just like the hacking minigame which I think is a nice parallel.

Both these additional states would have a significant impact on the core gameplay loop, and alter how levels have to be designed. Either because guards can exit their patrol route, or because getting caught is no longer an instant lose state.

The guards walk state is bugged. If they ever turn 180 degrees exactly, they won’t switch states.

## Win / Lose States

**GameManager, SM\_DetectionField, SM\_CardCache, SM\_EscapeDoor, SM\_SaveSpot, PC\_PlayerController, HM\_HackingManager**

Diagram

Description automatically generated

## Procedural Level Generation

Diagram

Description automatically generated

I didn’t have time to implement procedural level generation, but I did plan it out. My idea was to have a series of room prefabs which could be connected together like a jigsaw.

# Reflection

I feel positively about my game. I think it has some unique and fun gameplay aspects I’d like to develop more in my own time.

The game is lacking in many ways. The hacking and the stealth don’t mesh as well as I would have liked. It feels like you can skip many hacking segments. The game needs a few level design tricks to pair hacking and stealth together effectively. The game works best when the player needs to hide within or near a guards patrol in order to be in range to hack a certain camera. Using that trick over and over would make it boring though. The game is also lacking in feedback. The hacking minigame especially would feel more fun if the player got sound and particle effects.

I learnt to plan my code effectively, which helped reduce clutter both in my files, and in the scripts themselves. It also helped write code that’s can be re-purposed as I thought critically about the use cases of each script and identified overlaps. (E.g. The guard and security camera both need detection fields.) I also learnt about switch statements and interfaces which where both incredibly useful.

# Appendix

## Video Playthrough

<https://youtu.be/w4BfQ6MX61M>

## Code that wasn’t mine but is in the project

Proto\_SingletonPattern is my experiments with the advanced singleton design pattern we were shown in class. I didn’t write it myself.

In the script SM\_PatrollingGuard, there is a few lines where I used an equation I found online to calculate the angle the guard is supposed to rotate towards. I have marked in the script where this maths was used.