Technical Plan & Risk Assessment

Cameron Schneider & Conner Root
EGD-220-01
Project 2 Team 2
Sprint 6 - 04/08/2019

Table of Contents

Development

Game Engine

We will be using the 3D template provided by Unity. We've chosen this engine to develop our game due to the massive amount of community support, the large toolset, and the ability to implement 2D art assets within a 3D space, since *Inkantation* is set in a 3D world. There is also a good amount of resources within Unity's asset store what will allow us to build custom libraries for various mechanics, which will be explained later.

Finally, Unity also gives us the ability to port to many other platforms with relative ease should we decide that the platform is a feasible option to pursue.

Development Environment

The IDE Visual Studio will be used for creating and editing the scripts used in the Unity project. Unity connects to Visual Studio by default, and is a great development environment for many languages, including C#.

Version Control

Git is our chosen version control system. The repository is set up and active, with multiple members using it. As the designers begin to create worlds, we will start to use branching in order to help avoid merge conflicts, and we will also have the programmers act as support when Git-related questions arise.

Target Platform

Our target platform for *Inkantation* is going to be Windows, due to limitations with our current level of experience and our choice of controller: the Wii remote. Nintendo has a very harsh price for their Unity development kit, so that means our platform is limited to the ones that are able to use USB Sensor Bar devices, which is primarily Windows, and will be distributed through Steam.

Given more time experience, we would like to see this game realized in VR, which can span multiple platforms. Depending on the level of success, our Unity would allow us to port the game to mobile and removing the need of a Wii remote.

Mechanics and Systems

Mechanics

Drawing - Medium Difficulty

Players must be able to draw symbols in the world, which need to then be accurately recognized by the system. Players will be drawing using the ‘B’ button on the Wii remote, and will be drawing in the air in front of them. The hardest part will be accurately recognizing symbols.

Summoning Demons - Easy Difficulty

When the player successfully draws a symbol, a corresponding demon will be summoned. This will be an instantiated prefab that will have a certain behavior, whether it deals damage to the enemies, or gives the player a buff.

Demon Behavior - Easy Difficulty

Demons that deal damage will act as stationary towers, dealing damage to enemies when they get in range of the tower. This is relatively simple, only needing a particle effect and a trigger. Demons that provide player buffs will be harder to do, since these demons will disappear, but leave the player with a new advantage. We need to be able to have demon scripts interact with both player behavior and UI.

Enemy Behavior - Hard Difficulty

Enemies will be entirely AI controlled. We need to have a navigational system, and will be using Unity’s AI systems to do so. Additionally, enemies will have attack behavior and health, so we need to be able to precisely control when enemies attack, and where they can move. The hardest part will be the navigation.

Player Combat - Medium Difficulty

Players can fight the enemies with their demons and spray can. Demon summoning has already been covered. One of the buffs will turn the spray can into a flamethrower, allowing the player to damage enemies, but without this buff, the spray can can only stun enemies. The hardest part will be deciding on a control scheme.

Systems

World Kit - Easy Difficulty

The world kit system will be a collection of all prefabs, and possibly some basic level building tools to make level design and implementation fast.

Prefabs will be sorted into their respective folders, such as: environment, demons, enemies etc.

Cursor Stabilization - Hard Difficulty

The Wii remote IR data is extremely sensitive and jittery, so we have created a custom DLL library to interface directly with Windows in order to set the cursor position manually from within Unity. We can then use lerp algorithms and a predictive algorithm to smooth mouse movement.

Wii Remote Input - Easy Difficulty

Since the Wii remote is converted into Keyboard and Mouse input by the sensor bar hardware, There is little work that needs to be done to translate it to Unity controls. We only need to create a small wrapper class to map Unity keyboard and mouse input to the Wii controller buttons.

Controls are mapped as follows:

Wii Remote	Keyboard and Mouse	Use
IR	Mouse Movement	Control mouse cursor
A	Right Click	Toggle drawing mechanic on/off
B	Left Click	Draw when drawing mechanic is on
D-Pad	Arrow Keys	Player Movement
Minus (-)	Escape	Rotate the player left 90 degrees
Plus (+)	Enter	Rotate the player right 90 degrees
Home	Windows button	N/A
Power	Turn off Bluetooth	N/A

Nunchuk		
Z	Left Click	
C	Right Click	
Joystick	Arrow Keys	

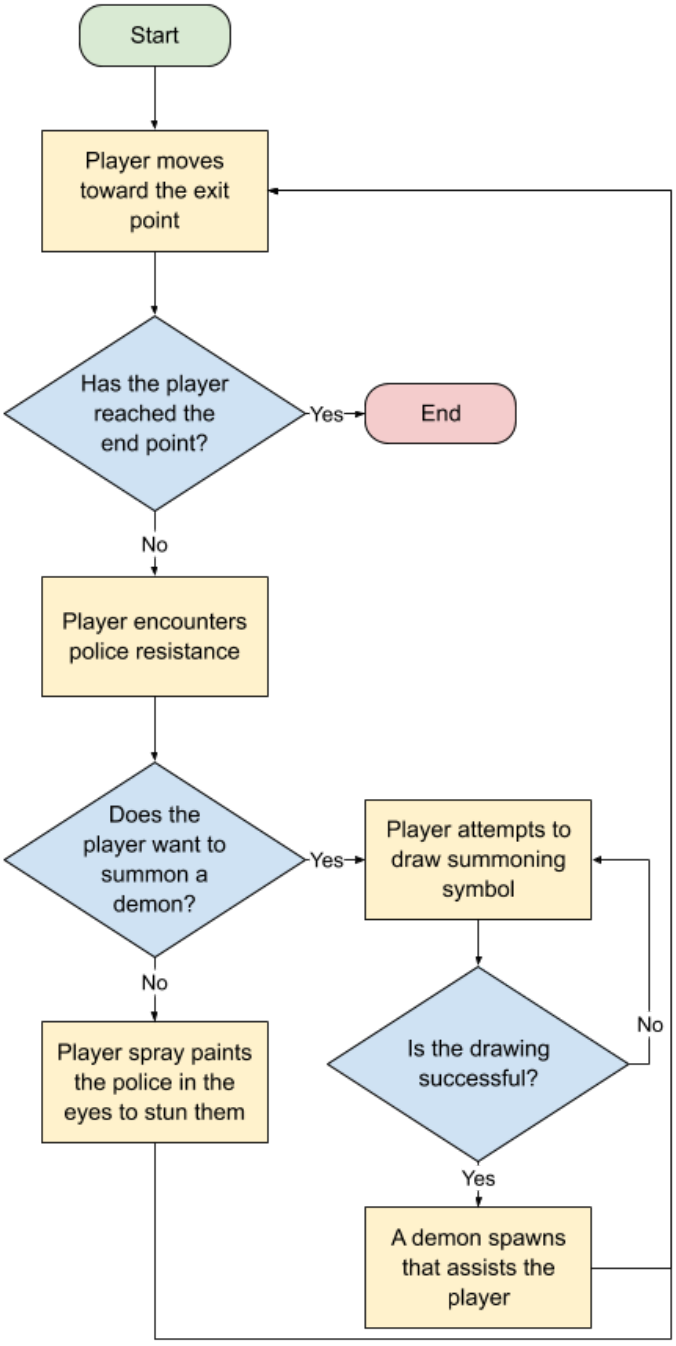
AI System - Hard Difficulty

The enemies will be controlled by an AI navigation mesh, which is conveniently provided by Unity, which means we will have plenty of resources for support. This is going to be the hardest system to manage and create, since we haven't had any experience using NavMeshes before.

Audio Systems - Easy Difficulty

We plan on having voice lines for the player, enemies, and demons, to give a bit more character to the game. The designers will be writing the voice lines, then they will be recorded on a quality microphone and edited as-needed. In-game, we must prevent overlap, so an audio queue will be needed.

Game Flow


Start
Player moves toward the exit point
Has the player reached the end point?
Yes
End
No
Player encounters police resistance
Does the player want to summon a demon?
Yes
Player attempts to draw summoning symbol
No
Player spray paints the police in the eyes to stun them
Is the drawing successful?
No
Yes
A demon spawns that assists the player

Hardware Requirements and External Assets

Hardware

The team decided to use the Wii remote as our controller, and in order to properly and efficiently interface with Windows, we either needed to create a custom driver, buy Nintendo's development kit, or find cheap hardware to easily handle the Bluetooth connection, and translate the data. Due to the price of the development kit, and the difficulty of creating custom drivers, we decided to find hardware to handle the connection and data. We found that the Mayflash DolphinBar sensor bar would be the best option. It is a custom sensor bar found on Amazon for $20.00, and it handles connecting the Wii remote over Bluetooth, then to a PC by USB. It also has 4 modes that the remote can be used by: mouse and keyboard, multimedia, game controller, and as a classic Wii controller. We decided to use the mouse and keyboard mode, as that is the most extendable with Unity. Another piece of hardware is also the controller itself, but those are simple to find.

External Assets

We used an external asset for the Gesture Recognizer, found on Unity's asset store. Credit to Raphael Marques for creating this asset. However, we did not simply download the asset and use it—we've extended its uses by using it as a library. Originally, the asset was only intended to work in a 2D space as a simple drawing canvas, but we extended its uses into 3D space. We also modified a few methods of how the asset recognizes drawings to more accurately reflect our needs.

Pipelines

Design Pipeline

The design pipeline is streamlined by a world building kit, created by the programmers. This kit will include Unity standard prefabs for environment assets, which will consist of Unity's primitive objects so that our artist doesn't need to create 3D models, our gesture recognizer (discussed above), and various 2D building textures that can be swapped out at will. The world kit will also include instructions and tools to assist in creating navigational paths for our AI enemies, along with their associated scripts and art assets. The design pipeline also includes assisting in the development of control schemes, which involves a lot of testing and debate over efficient and feasible controls. For level design, one designer comes up with a basic sketch and goal set, then provides documentation while the other designer takes the documentation, and uses the World Kit in engine to build the level. Overall, the designers have a lot of control over the world environment, projected player actions, enemy movement patterns, and control schemes.

Public Variables

Script	Variable Name	Purpose
EnemyMoveScript	float speed	How quickly the enemies move
	float maxDetectDist	How close the player has to be before the enemy moves towards them
	float stunDuration	How long the enemy is stunned
EnemyScript	float health	How much health the enemy has
	float hitDist	How far the enemy has to be to hit the player
	float timePerHit	The time between each hit
	float damage	How much damage the enemy deals
	float hitForce	How far the enemy pushes the player when they hit
GameManager	Text winText	The text object that the loss text is displayed
KekScript	float activationTime	How long Kek stays active when triggered

LevelChanger	Animator anim	
MalacodaScript	float lifespan	How long Malacoda is visible before disappearing
NextLevelScript	LevelChanger levelChanger	The scene's LevelChanger object
PlayerScript	Image healthBar	The health bar in the UI
	Image fireBuffIcon	The icon in the UI that tells if you have the fire buff
	float maxHealth	Player health
	float boostTime	How long buffs last
	float fireDamage	How much damage the fire damage does
	PostProcessScript post	The Post-Processing script
PostProcessScript	float maxDrawVignetteIntensity	The intensity of the draw-mode vignette effect
	float maxDrawDistortionIntensity	The intensity of the draw-mode distortion effect
	float drawVignetteTime	How long the draw-mode effects take to show up
	Color drawVignetteColor	The color of the draw-mode vignette
	Float maxHitVignetteIntensity	The intensity of the hit vignette effect
	float hitVignetteDuration	How long the hit vignette is displayed
	Color hitVignetteColor	The color of the hit vignette
SpawnerScript	float activationRange	How far the player must be before the spawner makes an enemy
	float spawnTimer	Time between spawns
StartGame	LevelChanger	Level changer object in the main menu
SymbolScript	float sprayDist	How far the player's spray reaches

Art Pipeline

All of the art assets will be created on a 1920 x 1080 canvas in Adobe Photoshop and Illustrator. Doing this will establish scale between the assets and not be needed in-engine. All asset needs will be documented in a list on the team Google Drive folder, created, then exported as `.png` files and uploaded to Google Drive as well as the repository. They will be named using the following naming scheme:

`Piece-Of-Game_What-It-Is.png`

Ex: `Char_Malacoda.png`

Placeholder/concept art will be preceded with "`placeholder_`" or "`concept_`"

From there, programmers will add the assets to the world kit for designers to use wherever they need. Designers will also be working closely with the artist to create cohesive documentation.

In-engine, we've decided to use the block-out technique with environment art. Designers will first set up the level with no art assets outside of Unity's default tools, then another team member (either artist, designer, or a programmer) will go through the blocked-out level and begin adding environment art. That way, we're able to have level assets to test, even if they don't have art.

Sprint Updates

Sprint 1

- Team came up with concepts based on our individual spring break assignment.
- Settled pretty quickly on *Inkantation* as our target concept.
- Programmers were able to prototype and test multiple possibilities for controllers, settling on the Wii remote or a mobile controller as the main possibilities.
- Rhys and Edison drafted rough documentation.
- Adam drafted documentation
- Cameron and Conner began researching what would be needed to have gesture recognition and various controllers.

Sprint 2

- Team settled on a new vision of *Inkantation*, which changed a few of the core mechanics.
- Team settled on the Wii remote as being the best option.
- Cameron built a testable version of the game, just using the Wii remote to draw recognizable symbols.
- Conner got drawing working in a 3D space, so we can draw on buildings.
- Edison, Cameron, and Ryan attended a QA session to test the game.
- Rhys continued iterating on documentation and possible gameplay changes.
- Conner iterated on the game flow document.

Sprint 3

- Ryan has been holding a lot of meetings, keeping development on track and organized.
- Adam got a few concepts for demons done, as well as his documentation.
- Conner was able to implement demon summoning, and made the last iteration of the game flow diagram.

- Cameron smoothed the Wii remote jitter based on QA results, and built another testable version of the game, this time to test movement and drawing in a 3D world.
- Edison and Rhys had disagreements on how movement should be handled, but came to a mutual conclusion.
- Rhys made some concessions for how drawing should be handled.
- Edison and Cameron attended another QA session.

Sprint 4

- Cameron got drawing in 3D, drawing in air, and strafe movement all functioning and implemented.
- Rhys sketched out a design for a level.
- Edison polished documentation and some smaller art assets for the game.
- Adam got more demon art, enemy art, a logo, and environment art done.
- Conner got both demons functioning, enemies functioning, and a start on player health.
- Ryan held a lot of meetings, and kept working on production and organization.

Sprint 5

- Adam finished up a few of the main art assets, and got some UI assets done as well.
- Cameron implemented the audio system, and got all of the voice lines recorded.
- Rhys and Edison have been collaborating on level design, and building levels in the engine.
- Conner got player combat working now, and also added Malacoda back in with his intended effect.
- Ryan has been keeping production documents up to date.
- Cameron got more UI cleaned up and implemented.

Sprint 6

- Adam finalized demon art as well as created environment assets and UI assets.
- Conner added in Unity's post-processing stack for visual effects and fixed multiple bugs that caused the game to crash.

- Rhys and Edison have been collaborating on level design, and building levels in the engine.
- Conner and Cameron added in multiple post-processing effects to offer visual feedback in-game.
- Ryan has been keeping production documents up to date.
- Cameron overhauled the UI and ironed out issues in the Audio System.