Technical Design Document for *A Price To Pay* Executive Summary

MJ's JRPGs

Micah Grande, Gerald Lappay, Jack Chen, Richard Estes, James Zheng, Patrick Noosaeng

Game Development Team:

- **Micah Grande**: Enemy AI: Enemy Spawning, Implemented Credits scene, Created Prefabs, Quality-of-Life Improvements
- **Gerald Lappay**: Audio Design: Voice Acting, Git Management
- **Jack Chen**: Enemy AI: Flocking Behavior and Tracking, Implementing Spawn Point and Treasure Art Assets
- **Richard Estes**: Audio Design: Music Composition
- **James Zheng**: Procedural Maze Generation, Implementing Ghost Art Assets
- **Patrick Noosaeng**: Documentation: Game Design Document and Technical Design Document

Game Overview:

A Price To Pay is a first person, Survival Horror, Stealth, Procedurally-generated maze game where a player traverses a maze to try and find treasure and bring it back to the spawn point without being caught / touched by enemies.

Technical Summary:

A Price To Pay has been developed over the course of a month by six people using the Unity Game Engine. The total cost to create the game is $0, using resources created in the Unity Engine and the Unity Asset Market. There is no planned revenue for the game and there are no plans for future releases in the series.

The game will be deployed for Windows, MacOS, and Linux platforms.

The minimum requirements include:

WINDOWS, MACOS, LINUX STANDALONE:

- Windows XP SP2+, MacOS X 10.8+, Ubuntu 12.04+, SteamOS+
- Graphic Card: DX9 (shader model 2.0) capabilities; generally anything made since 2004 should work

Equipment:

One MacBook Air 18', One MacBook Pro 18', One Acer Aspire, One Dell HPS 13, and One Lenovo Yogan N20, and Leapfrog Thinkpad computer, were used as the primary hardware platforms for game development and asset creation.

There is no cost associated with these devices as it applies to the creation of the game, as these are the personal computers of the developers.

Software:

All the software used for the development of *A Price To Pay* will be able to produce medium to high quality visuals, while still being able to deploy across different platforms.

Unity Community Edition and the Visual Studio IDE were used, which are free to use. Pages and Wordpad were also used, which are included with MacOS and Windows respectively.

Git was used to centralize the development of the game between each of the developers.

Evaluation:

Game Engine:

Unity was used because the 3D game could be created easily and be highly optimized, as well as deployed relatively quickly to Windows, Mac, and Linux devices without significant cost. Unity's services allowed our team to speed up the development process, optimize our game, connect with an audience, and achieve success

Target Platform:

This game will primarily be deployed to the Windows, Mac, and Linux user-bases, as the objective for this game is to quickly deploy a 2D game to these platforms though Unity. Being able to deploy across multiple platforms will make this a very efficient method for fulfilling that purpose.

Scheduling:

Development Plan:

The game was developed over a month at the availability of the six developers of the game:

Milestones:

- Implemented the Procedurally generated maze
- Implemented the first person player view with a flashlight
- Implemented Ghost spawning
- Implemented Ghost flocking behavior
- Implemented obtaining Treasure and bring back to spawn objective
- Implemented UI and mini-map

- Implemented Main menu, Credits, and Splash screen
- Implemented Unity Store Assets into game
- Completed Debugging
- Gold

Work Environment:

- Completed at SDSU and remotely from home

Levels:

Maze: One single, procedurally-generated level, consisting of the Price ghosts that chase the player, and a treasure at the opposite end of the map that the player must find and take back to the spawn point of the maze. There is only one spawn point in the entire maze.

Assets List:

- Enemies - Stalker Characters with Price face from: http://www.cs.sdsu.edu/faculty-and-staff/
- Maze
- Floor Segment - From Unity Store: https://assetstore.unity.com/packages/3d/environments/dungeons/floor-segment-20330
- A Piece of Nature - Treasure Chest - From Unity Store: https://assetstore.unity.com/packages/3d/environments/fantasy/a-piece-of-nature-40538

Script Analysis as it applies to Game behavior:

All Scripts were written in C#

- CloseGame.cs - Controls exiting of the game by using the Escape key
- Credits.cs - Controls the movement of the credits text

- Exit.cs - Unused script
- FpsMovement.cs - Controls player movement. Includes functions to rotate the character and the camera with the mouse, and to move the character with the directional keys
- GameController.cs - Calls MazeConstructor to obtain and spawn the procedurally generated maze, and controls game events, such as resetting the maze, and events that occur when the goal is reached
- GoalController.cs - Controls interactions with the Treasure item
- InstructionsToggle.cs - For toggling UI functions using keys
- LevelController.cs - Controls scene loading
- MazeConstructor.cs - The script responsible for generating the actual maze in a procedural fashion and determining the location of the spawn point and the treasure (The spawn point is always set at the bottom left, and the treasure is always set at the top right, and there is always a path between both). Also disposes the maze and creates a new one upon completing the maze or getting hit by an enemy.
- MazeDataGenerator.cs - Generates data to fill the maze
- MazeMeshGenerator.cs - Creates the meshes for the walls of the maze
- SpawnController.cs - Continuously checks if the player has reached the spawn point after obtaining the treasure, at which point it will reset the maze and restart the game
- StalkerMovement.cs - Controls the movements of the stalker characters, and controls their reactions when the player falls into their line of sight. Will also be responsible for controlling flocking movement

- StalkerSpawn.cs - Controls the spawn locations of the stalker characters. Will also control spawning of the larger stalkers when the treasure is found.
- TriggerEventRouter.cs - Helps sets up the collision event with the player and the treasure
- ToMenu.cs - Controls sending the player to the main menu upon pressing the Escape key

Script Complexity Analysis

- CloseGame.cs - **O(1)** - Calls a single function to exit the game upon pressing the Escape button
- Credits.cs - **O(1)** - While the script is applied to all of the credits text, it does not have to iterate through a list. As such, its functions run in constant time
- Exit.cs - Unused
- FpsMovement.cs - **O(1)** - No function requires iterating through a list of modifiable size. As the functions are solely meant to continuously control the movement of the character and the camera based on user input, the complexity should be constant
- GameController.cs - **O(1)** - No function requires iterating through a list of modifiable size. This script is solely designed to control important aspects of the game without needing to iterate through the list
- GoalController.cs - **O(1)** - No function requires iterating through a list of modifiable size. This script is meant to

- InstructionsToggle.cs - **O(1)** - No function requires iterating through a list of modifiable size. Each function operates in constant time
- MazeConstructor.cs - **O(n^2)** - The maze generation functions require iterating through each row AND column to place and find the start and goal positions, and therefore is based on the size of the maze.
- MazeDataGenerator.cs - **O(n^2)** - Has to iterate through each row AND column to iterate the maze array with data, and therefore is based on the size of the maze
- MazeMeshGenerator.cs - **O(n^2)** - Has to iterate through each row and column to generate Quads to generate the walls of the maze, and is therefore based on the size of the maze
- SpawnController.cs - **O(1)** - Does not need to iterate through a list while it checks to see if the maze needs to be reset because the player completed the level. When the player completes the level, the time taken is constant
- StalkerMovement.cs - **O(1)** - While this script is applied to all Stalker objects, it does not have to iterate through them. Each function runs in constant time
- StalkerSpawn.cs - **O(n)** - Spawns all of the Stalkers and their locations on the map at the Start(), and is therefore dependent on how many stalkers need to be generated.
- TriggerEventRouter.cs - **O(1)** - This event handler does not iterate through a list of modifiable size, and is only meant to set up the collision interaction between two objects, therefore it runs in constant time.

- ToMenu.cs - **O(1)** - Continuously checks to see if the Escape button is pressed, and if it is, the scene is shifted to the Main Menu scene