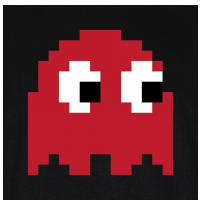
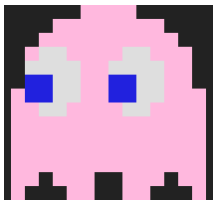


Pac Man Retribution

Matt Hudson and Song Vu Nguyen
Computer Graphics 1 - Spring 2019
Final Project Report



1. Project Description & Overview

Pac Man Retribution is a project built using Unity for the final project of CS 4610 - Computer Graphics 1. We created a Pac-Man style game with a twist: Pacman can shoot and kill enemies. At the start of the game, player will be spawned in a randomly generated maze where they need to collect pellets, defeat enemies, and acquire upgrades and powerups to survive. New enemies will spawn at a rate (which increases every level). There are four different types of enemies that spawn around the maze. Details about each type of enemy can be viewed in the about menu of the game (in short, there is a normal enemy, a fast enemy, a slow enemy, and a special enemy). Level increases every 20 seconds, which increments player & enemy stats. At certain levels, the player will receive a random upgrade, new enemies might spawn, and/or pellets regenerate throughout the maze. There are also powerups which spawn randomly around the maze which the player can collect. Each type of power up has a different effect on the player which is explained in detail in the about menu of the game. The goal of the game is to survive as long as you can and to get the best score. Players' high score will be saved and can be viewed in the main menu.

2. Methods and Implementation

The game was created using the Unity engine, which provided tools for creating and defining scenes, assets, and graphical settings (materials, lighting, cameras, etc.) that we learned about during the semester. While developing in Unity abstracts much of the lower-level implementation such as that which we used with OpenGL, GLEW, etc. it still utilizes those concepts and ideas in the construction of the game. Unity provided tools for us to develop the standard Model-View-Controller structure for an Object-Oriented application:

Model - resources such as prefabricated game objects (i.e. player character, enemies, pellets, maze components, etc.) were created in the Unity Editor and used to model the game. Some scripts also contained variables such as score, health, etc. and timers to model the game data and environment.

View - We used the Unity Canvas system to construct a static UI for the game (constructed as a 2-dimensional overlay on the screen). The game (i.e. the Maze, projectiles, enemies, etc.) exists in 3D and is viewed by a static camera looking down on the maze. The camera and UI combine to create the view for the game.

Controller - Unity uses C# scripts to control game actions, and we use many scripts to control various aspects of the game. The Unity Editor also implements buttons with On Click functionality, which we set up in the Unity Editor (i.e. change active game objects on the click of a button).

Below is a list of the features we implemented and our design process:

****Note - our scripts are hundreds of lines long, and it would be infeasible to go through the code like that in this report. We provide a brief overview of the design process, but feel free to ask us about specific implementation if necessary.****

1) Both of us had not utilized the Unity Engine much before, so we began by learning the editor and using online documentation to create some test scenarios and learn how to use the editor.

2) Maze Generation - for this section, we utilized an online free resource available on the Unity Store. The package contained some predefined assets and scripts to generate a random maze, and included a small game where you could move a ball around and collect coins. We modified those assets and the code to: a) change the style of the maze to a Pac Man-like style, b) change the 'goal' system in order to spawn pellets instead of coins, and c) changed the 'ball' to the player character (Pac-Man) and changed the movement physics to a static movement rather than rolling around the maze. We also changed the camera to be static overlooking the maze (instead of following the player around) and to edit the background and lighting system. We used the RollerBall class (formerly the class governing the player ball) to serve as our Player Controller.

3) Projectiles/Shooting: We created a small game object that is spawned in the direction the user is facing with a set velocity. Unity uses RigidBody components to detect collisions with other objects. We tagged walls and enemies with certain tags that are used in the controllers to perform certain actions based on what entity the projectile collides with. By default, at this point, projectiles simply were destroyed if they collided with a wall or enemy, or went out of bounds.

4) Enemies: We started off with one enemy (the Red enemy). We later expanded to 4 enemies:
Red (Bink) - medium speed, medium damage, medium health.
Orange (Cly) - high speed, low damage, low health.
Cyan (Swole) - slow speed, high damage, high health.
Pink (Agent H) - super high speed, extreme damage, practically invincible, but only lasts for a certain time.

The enemies are constructed using prefabs - by default, rendered as a cube that is texture mapped with a texture of their respective ghost. This is a way we incorporated texture mapping into our final project.

5) Enemy Spawning: Our Enemy class serves to control the spawning of enemies. It keeps track of timers and spawns enemies in a random location (determined by a formula using numbers generated by an instance of the Random class) when those timers exceed a certain value. That value differs for each type of enemy and decreases upon level up. At this point, enemies just spawned randomly and did not move. At the beginning of the game, only red enemies spawn, but at levels 5, 10 and 15, new enemy types begin to spawn as well.

6) Enemy Movement and Pathfinding: We configured the enemies to move up by default. We could have build a Nav Mesh (one of the pre-existing classes in Unity) to deal with Pathfinding, but we created our own custom system. An invisible game object is affixed to the enemy pointing forward (in the direction of movement). Once this invisible object collides with a wall, the enemy will attempt to turn left or right to find a new pathway. This allows them to dynamically and unpredictably move throughout the randomly generated maze.

As a side note, faster enemies sometimes phase through walls (this is due to the nature of collision detection and the construction of the pathfinding algorithm). We actually used this to our advantage; faster enemies phase through walls, and are deleted if they leave the map. This ensures that especially pink enemies only last a certain time in the maze.

7) Enemy Damage, Health and Death: Each individual enemy (build from a prefab) contains an instance of the EnemyProps class. It contains information pertaining to that specific enemy, including their health and damage. When a projectile hits an enemy, it reduces their health by the projectile's damage value (determined on the spawning of the projectile). If an enemy's health drops to 0 or below, the enemy is destroyed and removed.

8) Enemy/Player Fragmentation: We also designed an algorithm to make the game more enjoyable; the enemies and player "bleed" when they are hit or destroyed. This is achieved by the Fragment() method, which spawns a number of small cubes and adds explosive force to those objects.

9) Player Health and Death: The player spawns with 100 health at first. Health 'rolls down' similar to the game EarthBound, in that damage does not immediately reduce the player's health by that amount, but rather it rolls down, offering an opportunity for the player to heal or make a last stand. Once health reaches 0, the game effectively stops: the player explodes, all enemies explode, and other game functions cease. The final score is displayed and the player must go back to the main menu.

10) Lighting: We also wanted to include lighting systems (such as we developed in Assignment 3a) into the project. Unity offers different types of light sources (i.e. light source properties), and also for materials (the properties of rendered meshes or objects) to emit light (i.e. material lighting properties). Projectiles have a point light, and enemies have a spotlight in the direction of movement according to their color. Other materials such as the maze components, player, fragments, etc. emit different colors of light as well.

11) Score: Each game has a score value which increases in many ways. Collecting pellets earns a small value (10), but collecting all pellets within a 10 level period (when they respawn) grants a large bonus. Defeating enemies also increases the score based on the type of enemy: later enemies grant a higher score value. Upon player death, their score is saved to the highscores.txt file. This is used to populate the high scores menu later. The score display also flashes to indicate the game is finished.

12) Score Title: a small utility function is that the "score" object on the UI changes color as the player plays the game; it doesn't provide a practical purpose but it adds to the presentation. A seperate class Colorer uses Color methods to dynamically change between colors.

13) Player/Enemy information displays: The other main component of the UI is displaying player and enemy statistics. Player health is shown with the bigger number being the current health and the smaller number as the max health. The display is green when health > 50%, yellow between 50% and 20% and red when health < 20%. The damage display shows the damage each projectile deals. Each enemy has stats for its health and damage as well. Each display also has an enlarged model of its prefab to display - a fun easter egg is that projectiles can destroy these models!

14) Time display: A simple timer is used to display time between levels. When it reaches 0, it triggers a level up and resets the timer to 20 seconds.

15) Leveling System: The leveling system is what makes the game dynamic and challenging as the game goes on. Each level:
- Player health is increased by a random amount between 5-25, up to a maximum of 500. The player is healed by this amount as well.
- Player damage is increased by a random amount between 0-3, up to a maximum of 100.
- Active enemies health and damage are increased by a fixed amount indefinitely.
- Active enemies spawn times decrease by a fixed amount until a minimum value is reached.

In addition, every 5 levels, the player is awarded a random upgrade of the 7 available (see 16. Upgrades) and is healed for 25% of their maximum health.
At level 5, orange enemies begin to spawn. At level 10, cyan enemies begin to spawn. At level 15, pink enemies begin to spawn.
Every 10 levels, the pellets in the maze regenerate.

16) Upgrades: There are 7 available upgrades that are awarded at random every 5 levels. Each upgrade can be gained only once per playthrough. These upgrades are:

1 - Damage: permanently add 50% of the current player damage to player damage.
2 - Powerup: make powerups last 50% longer (see 17. Powerups).
3 - Piercing: Projectiles pierce through enemies (no longer destroyed upon contact).
4 - Phase Shift: Projectiles phase through walls (no longer destroyed upon contact).
5 - Shield: Adds an aura effect and reduces enemy damage by 30%.
6 - Speed: Player moves twice as fast than usual.
7 - Rapid Fire: Player shoots 25% faster as usual (smaller reloadTime).

Available upgrades are displayed as greyed out (transparent). They light up (become opaque) when they are acquired.

17) Powerups: There are also 7 available powerups that can spawn randomly in the maze. After a certain time, a random powerup (1-6) is spawned in a random location. They can be collected in a period of time before they disappear. These powerups are:

1 - Damage x2: Double damage for 10 seconds.
2 - Score x2: Double score for 15 seconds.
3 - X shot: projectiles now spawn in all four directions regardless of player orientation for 10 seconds.
4 - Heal: heals for 50% of maximum health.
5 - Power: Similar to the power pill from the original Pac Man, makes the player invulnerable and destroys enemies upon contact with the player for 10 seconds.
6 - Full Auto: decreases the reload time to a very small amount. Makes the player shoot incredibly fast by holding down Space for 10 seconds.
7 - Ultima: This powerup only starts spawning at level 15. It also uses its own separate spawning timer and lasts longer before disappearing. It instantly kills all current enemies in the maze, but it is also rare and spawns less than other powerups.

Powerups can be distinguished by their aura (achieved using a Halo effect), which is colored according to the powerup. Available powerups are displayed as greyed out (transparent). They light up (become opaque) while the powerup is active. For Heal and Ultima, the displays light up for 5 seconds because they are not timed.

The Powerup upgrade extends the duration of powerups 1, 2, 3, 5 and 6 by 50%.

18) Progression: This is the goal that is achieved through a combination of leveling, upgrades, powerups and different enemies. These elements combine to make the game both harder and more fun to play as the game goes on. Powerups can also be combined with upgrades to make for some crazy combinations (ex. phase shift + x shot + full auto). We feel this makes the game much less trivial and a polished project.

19) Main Menu: Aside from the game mechanics, the main menu is where the player starts upon starting the game. They can Play the game, which takes them to the game scene. It has an intermediate 'loading' screen that is displayed while the game scene is loading. The Highscore screen shows the top 5 scores taken from the highscores.txt file (sorted by score value). The about screen contains information about the project and tutorials on how to play the game, and information about enemies, powerups and upgrades. They can also exit the application from here.

20) Music and Sound Effects: Music was taken from online video sharing sites such as YouTube. Since this is just a student project and this is not intended for profit of any kind, we aren't as concerned with constructing original music (after all, we're using the Pac Man characters after all). However, custom sound effects were created for the game, including shooting, damage, startup, etc. Only one sound effect (the gun sound effect) was taken from soundbible.com.

21) Builds: Unity allows us to create builds of the game, which can be run as standalone programs on Windows, Mac OS X and Linux distributions. We were able to build for all three of those platforms and include them in this repository.

We managed our project progress through Unity Hub, which is essentially the same as GitHub. The appropriate page is found here: https://developer.cloud.unity3d.com/orgs/hudso1898/projects/pac_man_retribution and details the project's history as we collaborated. After the final project was complete, we transfered the source code and builds for version 1.0 to this GitHub repository.

As always, there is always room to build upon our current implementation:
- Fragmenting enemies causes a lot of graphical strain, and slows the game down later on.
- We could add an option for players to enter a name associated with their high score.
- The pathfinding algorithm does not work perfectly, in that it cannot detect gaps in the middle of long pathways (i.e. the enemies can only turn when they hit a wall). We could also fix the phasing of enemies and find another method of limiting the existance time of pink enemies.

Ultimately, we feel as though this project fulfills the requirements because while it uses Unity and some borrowed assets, the vast majority of code, resources, and building of the game was done by us. It took many hours to implement just one of these features, so this project represents the culmination of about three week's worth of software development.

3. Team Members and Role Assignments

Our team consisted of two members, Matt Hudson and Song Vu Nguyen. Below are assignments and a breakdown of who developed what feature:

Matt Hudson - main menu graphical design, filling in information in the about screen, game resource design, implementation of game mechanics (pathfinding, enemy spawning, damage, health, powerups, upgrades, shooting, music, sound effects), custom sound effects, game UI design and controller, level up system, enemy fragment system, enemy design, miscellaneous utility functions for game, editing and revising presentations and reports.

Song Vu Nguyen - brainstormed idea for the project, Unity help and introduction, UI design (main menu and utility scenes), implementing loading screen and highscore system (with file operations), scene changing, writing initial presentations and reports, testing game mechanisms and bugfixing.

4. Source Code, Executables, and Report

The source code (including Unity resources, scripts, and unity project details) are uploaded in the "source" directory of a GitHub repository created to host the final version of the project. Built executables for Windows, Mac OS, and Linux are also included in the "builds" directory. For the purpose of the in-class and video demonstrations, the Mac build will be used, as both of our machines run OS X. A readme on the repository also serves as the report, but this PDF document will also be uploaded there as well, and turned in as the hard copy on Thursday May 16.

The GitHub repository is located here: https://github.com/hudso1898/Pac-Man-Retribution . All of this material will also be uploaded to Canvas in a compressed zip file.

5. YouTube Demonstration

A YouTube demonstration of the application can be found here: https://www.youtube.com/watch?v=OSIW542rPW8

You could also find it by searching “Matt Hudson Pac Man Retribution Demo” on YouTube; it is a public video.

The demo video is a brief overview of a sample playthrough of the game. You are more than welcome to download the game and play for yourself!