UFO Game

Austin Matias

Edward Hicks

Jacob Schumacher

Table of Contents

**1. Project Definition (**100 - 200 words**)** – *Group responsibility*

* Why (it is needed)
  + UFO Game is a game designed for user entertainment and competition. The genre mashup we are going for is rarely attempted and as such our game is serving a community that does not have many games to play.
* What (is the goal of the project)
  + The goal of this project is to develop a roguelike/metroid-vania genre mashup with a high replayability factor that will be entertaining and encourage players to compete against each other for high scores.
* How (how will it be achieved)
  + We will be achieving the above goals mainly through leaderboards and procedural generation, as well as a short game cycle. The leaderboards will encourage competition both between two players, as well as a single player as they try to beat the current high score. The procedural generation will keep the game fresh and varied across multiple play sessions, and the short game cycle ensures that a bad run does not take too much of the player’s time, and they can try their luck again quickly.

**2. Project Requirements** – *Group responsibility*

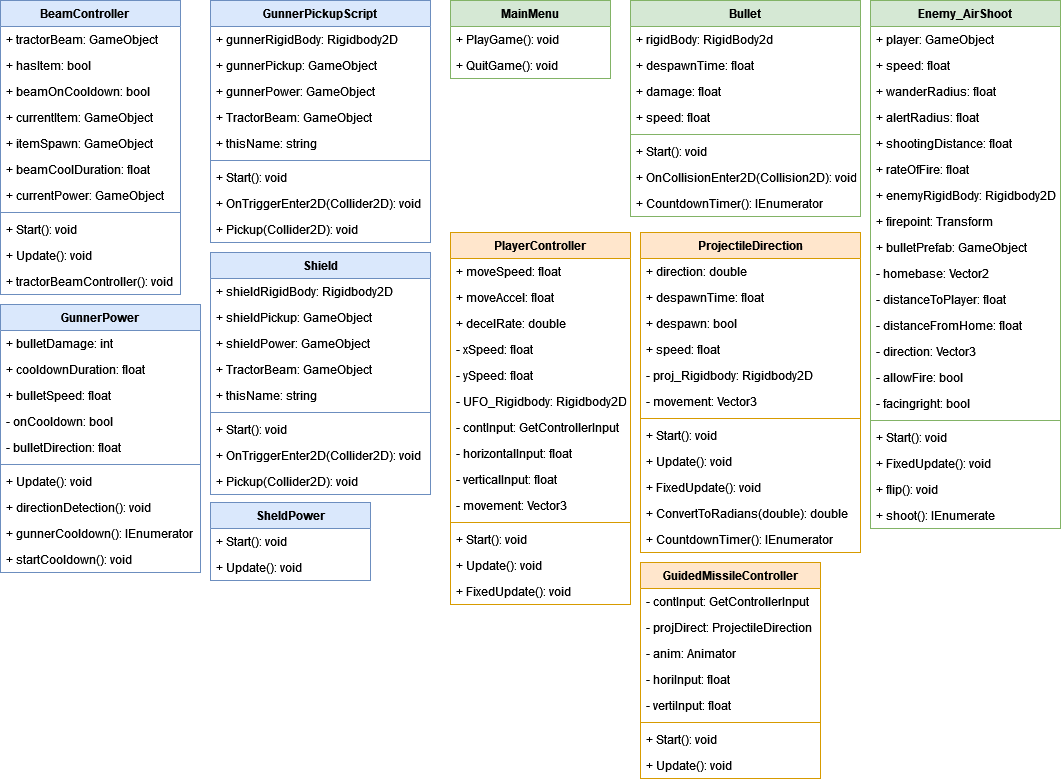
* Functional
  + A functional run-through of the game, once complete, will consist of the player starting in the starting room, and navigating through a level consisting of environmental hazards and roaming enemies to reach the destination. Throughout their journey, the player will pick up items they must use to mitigate the hazards and enemies to allow them to safely reach the end. If they fail to reach the end without sustaining too much damage, they encounter a game over and must restart.
* Usability
  + User interface
    - The user will be able to tell their current health, pickup, score, and progression through the game in the heads-up display.
  + Performance
    - The style of game we are building lends itself to excellent performance regardless of hardware.
* System
  + Hardware
    - We are basing our game on old 8-bit style games, and therefore hardware requirements will not be very intense at all, and any computer running windows should be able to launch the game with no problem
  + Software
    - No additional software will be needed to run our game. Just the game executable itself
  + Database
    - We are planning to store high score information with a simple cypher. This is currently the only persistent data we are planning to have
* Security
  + Since we are not hosting high score data on a sever, the high score list is subject to user manipulation. Our hope is that by using a custom file extension, as well as a basic cypher to obfuscate the scores, it wards off some people from simply changing digits.

**3. Project Specification** – *Group responsibility*

* Focus / Domain / Area
  + Our game is trying to reach players who enjoy old school style graphics, as well as those who enjoy either of the two genres our game is fusing together. Those genres being the roguelike and the metroid-vania genres.
* Libraries / Frameworks / Development Environment
  + Unity 2019.3.13f1
  + Aseprite v.1.2.39
* Platform (Mobile, Desktop, Gaming, Etc)
  + Windows PCs
* Genre (Game, Application, etc)
  + Roguelike/Metroid-vania Game

**4. System – Design Perspective** – *Group responsibility*

* Identify subsystems – design point of view

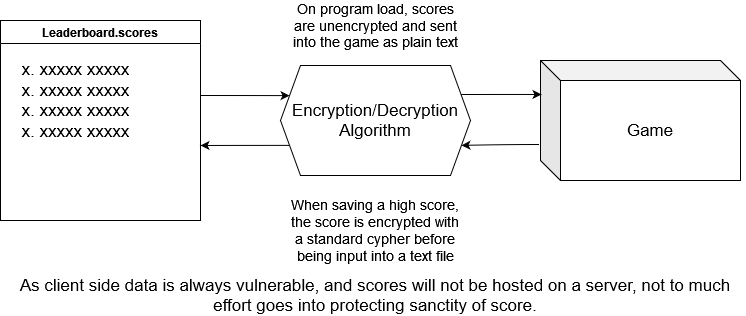


* Sub-System Communication (Diagram and Description)

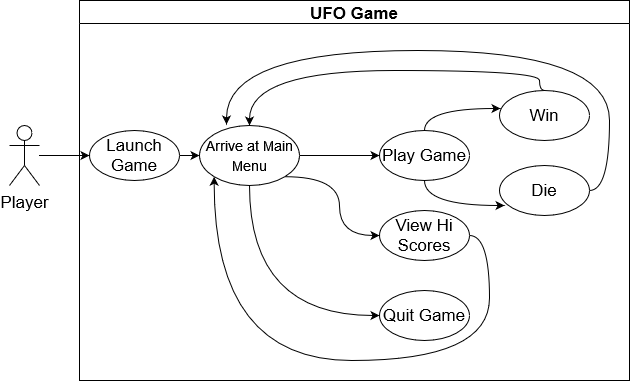
Diagram

Description automatically generated

* + Controls
    - Controls will be done through unity
      * Gives user ability to remap
    - 8 directional movement/item usage
    - A button to drop items
    - A button to active tractor beam/pick up item
  + I/O
    - Input will be done through a keyboard
    - Output visually through screen as well as audibly through speakers/headphones/etc
* Data flow and Storage



* Overall operation - System Model



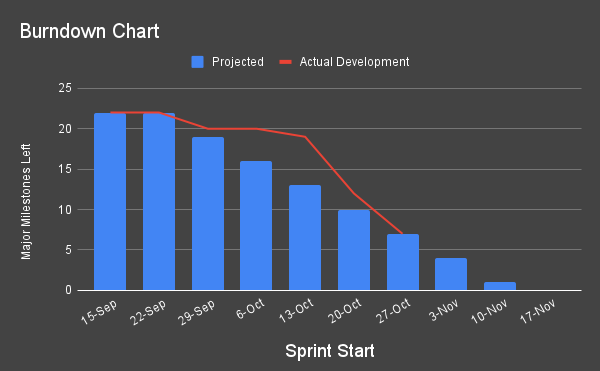
**5. System – Analysis Perspective** – *Group responsibility*

* Identify subsystems – analysis point of view
  + Scoring
    - The system that calculates the score as the user plays through the game.
  + Data storage
    - Encrypting, decrypting, and storing the high scores of the player.
  + Item pickup and Usage
    - The pickup and usage of the 9 items that make the gameplay in the game function.
  + Procedural Generation
    - The random level generation that adds replayability to the game.
* System (Tables and Description)
  + Testing Procedures
    - Each individual member progresses on developing their own subsystem.
    - As a new feature or major milestone of a subsystem is reached, the other members play around, and bug test the subsystem.
    - Since these members haven’t worked on the subsystem at all, they are sufficient bug testers.
  + Process models
    - The process model we are using is an Agile method, with scrums of 1-2 weeks in length. During each scrum, tasks are assigned, and functionality is built in a way similar to a RAD (Rapid Application Development) model.
      * Each person works on their own subsystem, and all of them come together once they are built up enough.
* Algorithm Analysis
  + Big O of main system
    - The big O of the game loop is going to be the same as unity’s built-in update method. All the subsystem complexities described below will have their big O described in a way that does not factor in unity’s own update big O.
  + Big O of sub systems
    - Scoring
      * Our scoring algorithm should run in constant time with the game engine, with a time bonus that ticks down the longer the level took, and points awarded for each enemy defeated.
    - Data Storage
      * The complexity of data storage is going to depend on the cypher used. However, the plan is to use a simple cypher, hopefully no more than O(n2) with n being the length of the string to encrypt/decrypt.
    - Item Pickup and Usage
      * Item pickup and usage revolves mainly around unity’s update method. Per frame however, this subsystem’s complexity is O(1).
    - Procedural generation
      * Details about the procedural generation’s big O are unknown at the current moment.

**6. Project Scrum Report -** *Group Responsibility*

* Graphical user interface

  Description automatically generatedProduct Backlog (Table / Diagram)
* Sprint Backlog (Table / Diagram)
* Burndown Chart



**7. Subsystems**

**7.1 Subsystem 1** – Scoring – *Austin Matias*

* Initial design and model
  + User gets a score for depositing items, and for the speed at which they complete the level.
    - Incentivizing dropping off items and quick completion.
  + Score master script, tracking variables in the level and using those changes to add to a score.
  + High score that was stored in plain text in the game’s folder
  + Illustrate with class, use-case, UML, sequence ..... diagrams
* Refinements over the course of the project
  + Score conditions were refined
    - Now the factors affecting score are player health remaining, enemies killed, time, and number/speed of items dropped off.
    - We are hoping these new factors encourage speed, skillful play, and a tactical mastery of which pickups are worth keeping and which are worth turning in quick.
  + High scores are now stored in Unity’s PlayerPrefs which, on Windows, is stored in the registry, meaning it’s safe from being messed with by players.
  + We will attempt to have a classic arcade style top 10 high score system instead of just the one, time given.
  + Change to the coding approach
    - Listed further down.
  + The new method will be running in sync with Unity’s update method, and other than that will have a constant running time.
  + Refined design (Diagram and Description)
* Scrum Backlog
  + Initial Development of this system took place between 21 Oct and 28 Oct 2022
  + Adjustments and tweaks are continuously made on a case by case basis
* Coding
  + C# Scripts
  + New coding approach involves a C# Script to handle all loading/resetting/adding of points
  + Event Listener
    - Listening for an event to reset points/load points/add points
  + Addition of points is then done in any other script
    - Points can be added simply by calling the event, and this can be done from anywhere. Making it very easy to assign new ways to get points and adjust how many points are given.
* User training
  + Each enemy type gives a unique number of points when defeated
    - Will update with specifics after balance testing the game
  + Finishing the level quickly gives you a time bonus
  + The amount of health you finish the level with gives you a set number of points per life point remaining
    - Will update with specifics after balance testing the game
  + Turning in pick-ups will give a set number of points. Each pickup has a unique number of points it gives, but as the player takes longer to turn in items, these point values decay.
    - Will update with specifics after balance testing the game
  + There is a bonus for turning in all 8 pick-ups before completing the level.
  + After finishing a level, if you achieved a high score, you will be able to enter your initials to have your score posted to the high score chart.
* Testing
  + We’ve been testing the scoring by assigning our enemies a point value, and killing them to ensure the point values are triggered correctly. This is the method we will use to test the functionality of the high scores as well.

**7.2 Subsystem 2** – Name 2 - *Individual responsibility*

* Initial design and model
  + Illustrate with class, use-case, UML, sequence ..... diagrams
  + Design choices
* Data dictionary
* If refined (changed over the course of project)
  + Reason for refinement (Pro versus Con)
  + Changes from initial model
  + Refined model analysis
  + Refined design (Diagram and Description)
* Scrum Backlog (Product and Sprint - Link to Section 6)
* Coding
  + Approach (Functional, OOP)
  + Language
* User training
  + Training / User manual (needed for final report)
* Testing

**7.3 Subsystem 3** – Name 3 - *Individual responsibility*

* Initial design and model
  + Illustrate with class, use-case, UML, sequence ..... diagrams
  + Design choices
* Data dictionary
* If refined (changed over the course of project)
  + Reason for refinement (Pro versus Con)
  + Changes from initial model
  + Refined model analysis
  + Refined design (Diagram and Description)
* Scrum Backlog (Product and Sprint - Link to Section 6)
* Coding
  + Approach (Functional, OOP)
  + Language
* User training
  + Training / User manual (needed for final report)
* Testing

**7.4 Subsystem 4** – Name 4 - *Individual responsibility*

* Initial design and model
  + Illustrate with class, use-case, UML, sequence ..... diagrams
  + Design choices
* Data dictionary
* If refined (changed over the course of project)
  + Reason for refinement (Pro versus Con)
  + Changes from initial model
  + Refined model analysis
  + Refined design (Diagram and Description)
* Scrum Backlog (Product and Sprint - Link to Section 6)
* Coding
  + Approach (Functional, OOP)
  + Language
* User training
  + Training / User manual (needed for final report)
* Testing

**8. Complete System** – *Group responsibility*

* Final software/hardware product
* Source code and user manual – screenshots as needed - Technical report
  + Github Link
* Evaluation by client and instructor
* Team Member Descriptions

***This is just a guide, and use it to create/improve your report. Feel free to add sections. You are responsible for your own subsystem/s, not other members. You have to contribute to the team’s goals and objectives, and develop your subsystem/s, write your documents and slides.***