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 metroidvania game 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, 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 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*

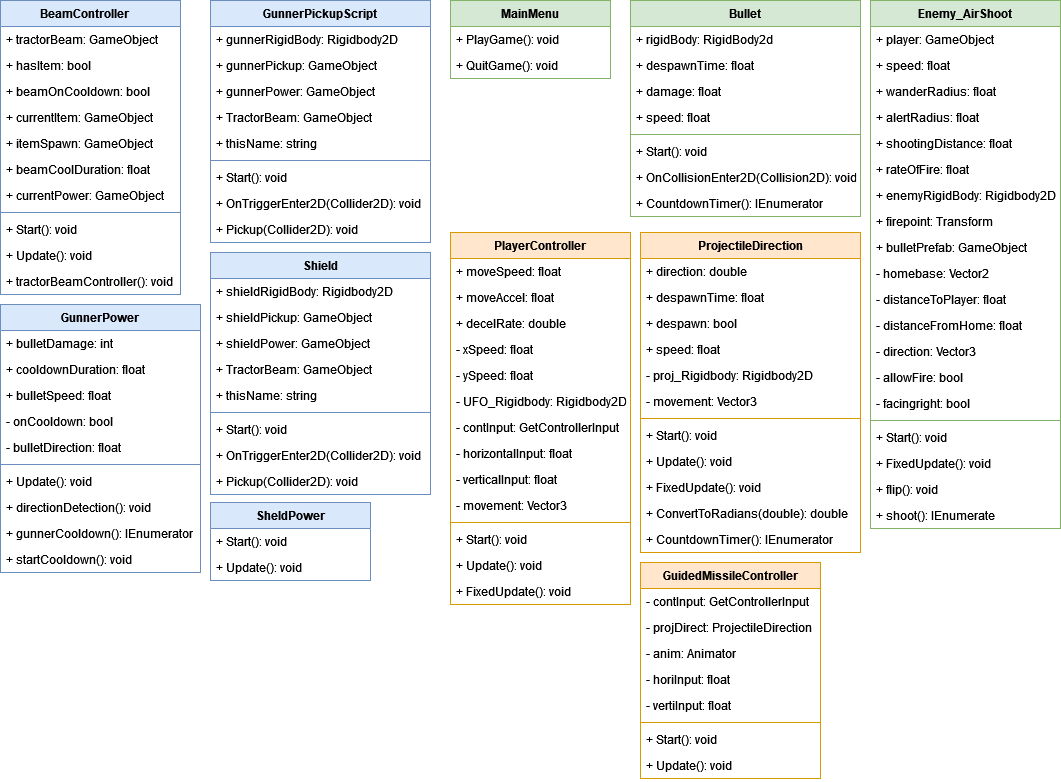
* Functional
* A functional run-through of the game, once complete, will consist of the player starting in the starting room, and navigating through a level consisting of environmental hazards and roaming enemies to reach the destination. Throughout their journey, the player will pick up items they must 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, time, and score.
* 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
* No database is required for our game. All information will be stored using Unity’s built in PlayerPrefs functions.
* Security
* Unity’s PlayerPrefs functions are stored in the registry on windows, and since that’s the only thing we are persistently storing, we don’t need to worry about any other kind of security.

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

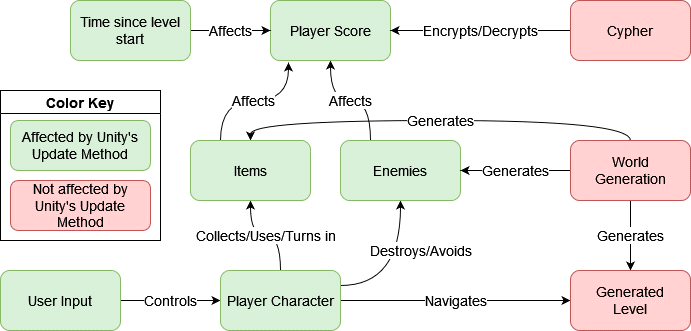
* Focus / Domain / Area
* Our game is trying to reach players who enjoy old school style graphics, as well as those who enjoy metroidvanias.
* Libraries / Frameworks / Development Environment
* Unity 2019.3.13f1
* Aseprite v.1.2.39
* Platform (Mobile, Desktop, Gaming, Etc)
* Windows PCs
* Genre (Game, Application, etc)
* Metroidvania Game

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

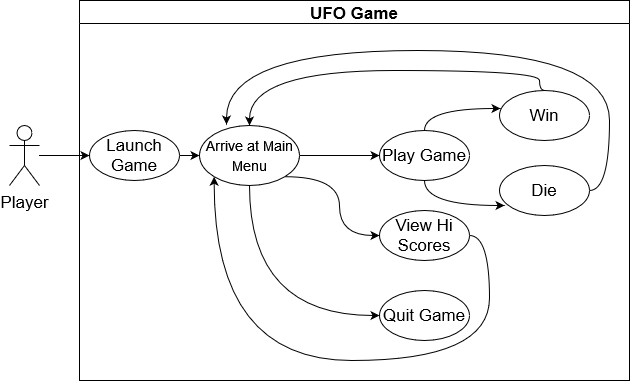
* Identify subsystems – design point of view



* Sub-System Communication (Diagram and Description)



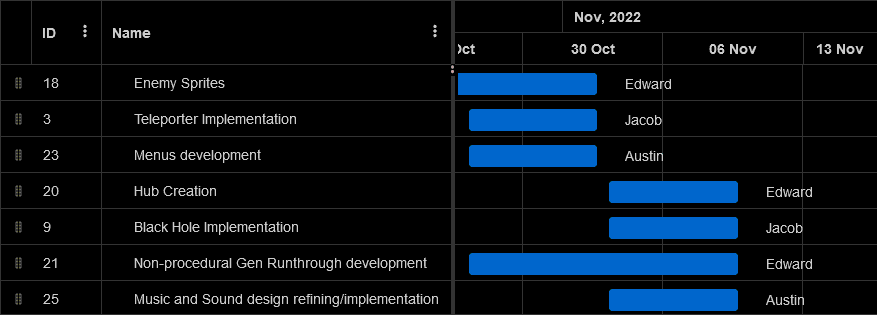
* Controls
* Controls will be done through unity
* Gives user ability to remap
* 8 directional movement/item usage
* A button to drop items
* A button to active tractor beam/pick up item
* I/O
* Input will be done through a keyboard
* Output visually through screen as well as audibly through speakers/headphones/etc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

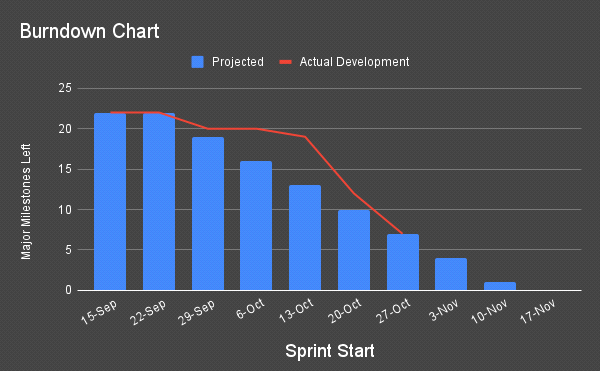


**5. System – Analysis Perspective** – *Group responsibility*

* 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
* Data storage complexity is based off of Unity’s PlayerPrefs, and runs in constant time.
* 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).

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

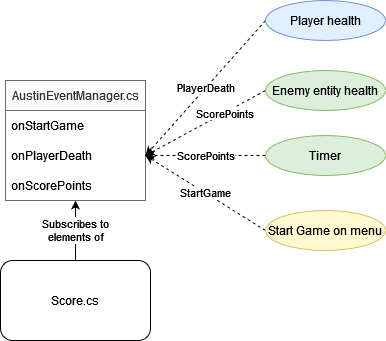
* Product Backlog (Table / Diagram)
* Full Gantt chart available online at <https://www.onlinegantt.com/#/gantt>
* Load the attached .gantt file.
* 
* Sprint Backlog (Table / Diagram)
* Can be found in our closed issues tab of the github page
* [https://github.com/jgschuma/UFO-Game/issues?q=is%3Aissue+is%3Aclose HYPERLINK "https://github.com/jgschuma/UFO-Game/issues?q=is%3Aissue+is%3Aclosed"d](https://github.com/jgschuma/UFO-Game/issues?q=is%3Aissue+is%3Aclosed)
* Burndown Chart



**7. Subsystems**

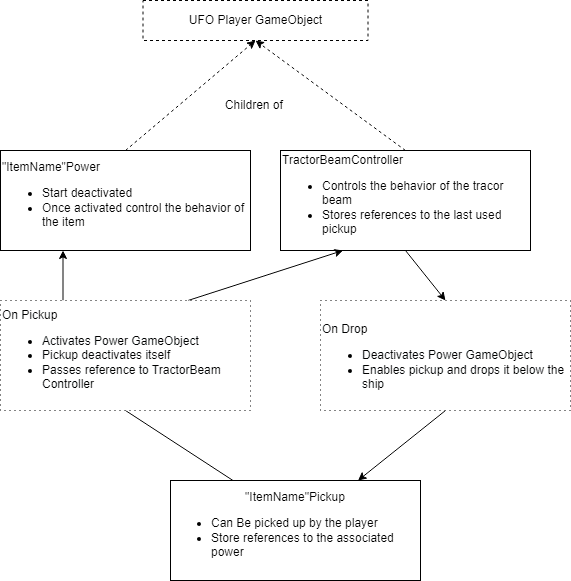
**7.1 Subsystem 1** – Scoring – *Austin Matias*

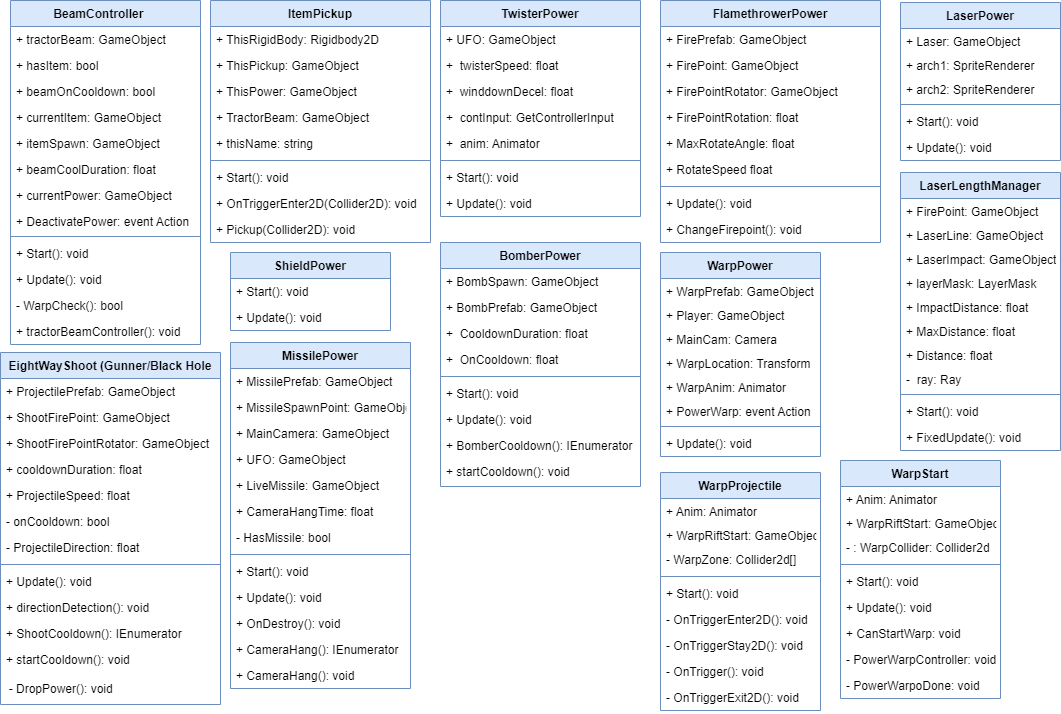
* Initial design and model
* User gets a score for depositing items, and for the speed at which they complete the level.
* Incentivizing dropping off items and quick completion.
* Score master script, tracking variables in the level and using those changes to add to a score.
* High score that was stored in plain text in the game’s folder
* 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** – Items and Tractor Beam – Jacob Schumacher

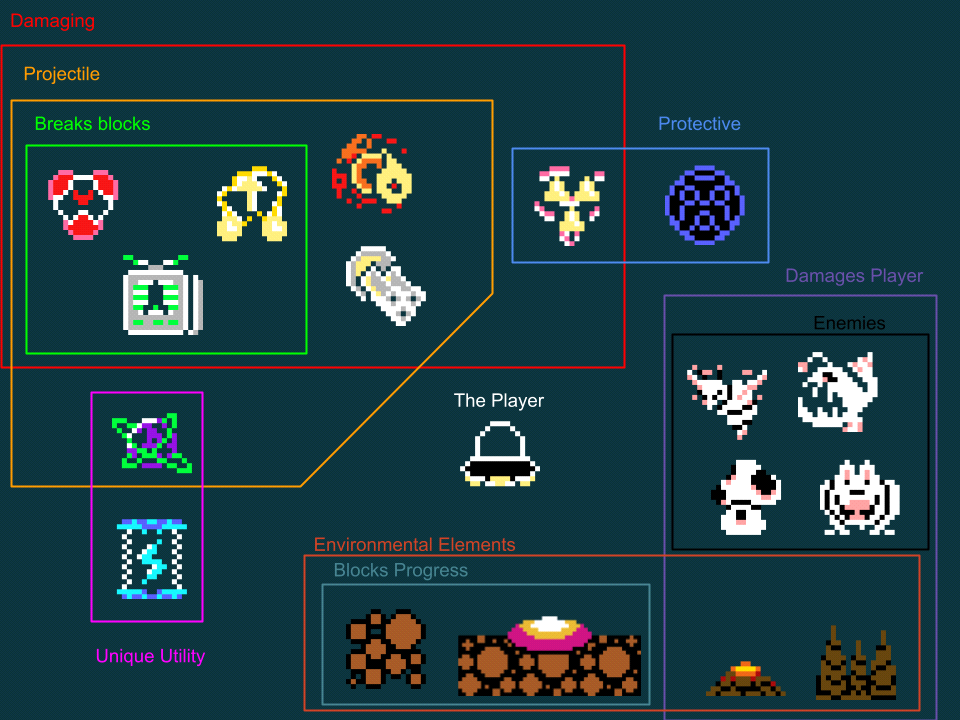
* Initial item design
* Item list and description
* No Power (Dash) – When not holding a power, the player can press the fire button to perform a short dash, increasing speed, does not apply invulnerability.
* Bomber – Pressing the fire button will allow the player to drop a bomb projectile. When the bomb collides with an enemy or the environment it will explode dealing damage to enemies and destroying breakable obstacles.
* Black Hole – Pressing the fire button will shoot a projectile in one of eight directions (up, up-right, right, right-down, down, down-left, left, left, up), after a short timer, the black hole will activate sucking in all enemies and dealing damage.
* Flamethrower – While the flamethrower is active, a nozzle will be displayed below the ship, that rotates towards the left and right based on horizontal player movement. The nozzle will have about a 180-degree range where it can aim. When the fire button is held, a stream of flames will be shot out in the direction of the nozzle damaging enemies.
* Gunner – While the gunner is active, pressing the fire button will fire a bullet in the direction the player is moving, these bullets will damage enemies
* Guided Missile – While the guided missile power is active, pressing the fire button will spawn a guided missile and lock player movement. The camera will follow the guided missile and using the movement buttons will instead control the missile. When it contacts with an enemy or the environment it will explode. This explosion will damage enemies and breakable obstacles.
* Laser – Pressing the fire button will allow the player to fire a beam straight up. This beam will damage any enemies it touches and destroy breakable obstacles.
* Shield – Holding the fire button will activate the shield, displaying a shield around the player, providing invulnerability to enemy bullets and attacks.
* Twister – While moving and pressing the fire button, the player will begin spinning rapidly, their movement will become locked and they will bounce off the environment while the spinning lasts, during which time any enemies will be damaged and sent flying.
* Warp drive – When holding the warp drive, the player can press the fire button to teleport back to start. If the item is dropped, the pickup will have a warp rift which the player can interact with to teleport back to start. This can be done while not holding the warp drive, allowing the player to bring other items through the rift. The start area has a rift that is only active when the warp pickup is active that allows the player to teleport back to the warp pickup.
* Each item would consist of two major parts
* The Item Pickup – The object that exists within the level and when picked up will allow the player to use the associated power
* The Power – The power is a GameObject attached to the player that controls the behavior of that power, it will be inactive until it’s associated pickup is selected
* The player will only be able to hold one power at a time
* The player would use a tractor beam to pull items up towards the ship
* The tractor beam has a cooldown after picking up and dropping items to prevent accidentally picking up or dropping an item immediately.
* Below are attached Diagrams for the class. Some of the Items have not been included yet as the scripts have not yet been written.



* Refinements
* The biggest refinement that has been made is that every pickup has been reworked to use a singular script to make the code re-useable.
* The pickups have all been turned into prefabs so that they can be easily moved to different scenes.
* Item Changes
* Gunner – The gunner will now fire a stream of bullets at a much faster pace, imitating a gatling gun.
* Guided Missile – The guided missile will be able to be detonated early by pressing the fire button after its creation. (This change has not yet been implemented yet)
* Scrum Backlog
* The initial concept development for the system was created during our first couple scrums early in the semester around late August or early September
* Coding
* All the scripts used in the creation of the item system are written in C#. The game objects and scripts are being used in Unity.
* All pickup items are handled by the ItemPickup script.
* The TractorBeamController script will hold a reference to the current power so that the relevant pickup could be dropped, and the power deactivated when the player activates the tractor beam again.
* Some of the scripts do utilize events, for example when an item is dropped an event is sent out to disable the cooldown of the active power to prevent bugs.
* Warp Pickup – The warp Pickup starts inactive, then is swapped for the active version (the one with the warp rift above it) when the item is dropped. The power itself will allow the player to teleport back to start by instantiating a prefab to cover the screen until the teleportation is finished.
* User Training
* The player will start with no item and be able to use a dash to quickly get around
* The player then will be able to search the level for items.
* Upon finding an item they can use the tractor beam to pull an item up to the ship and collect it.
* This will trigger a brief cooldown period where the item cannot be dropped.
* The player will then be able to use the power the pickup provides (shield, gunner, etc) until they press the tractor beam button again to drop the item.
* Dropping the item also triggers a short cooldown to prevent the player from accidentally picking up the item again
* The player will be able to sell their powers for points by dropping them in the mothership at the start of the level. This will disable the item for the rest of their run.
* Testing
* Testing for the items has consisted of the whole team experimenting with them to try and find unintended outcomes.
* This method allowed us to uncover bugs such as picking up multiple items at the same time (which would effectively delete one of the items).

**7.3 Subsystem 3** – Object Interactions - *Edward Hicks*

* Initial design and model
* Every object in the game interacts with each other in unique ways and needs to have behavior prepared for every scenario
* To simplify the process, several generic subsets of items were created to help divide objects by their behavior (ex: things that get shot by the player and enemies are Projectiles, player weapons that break blocks are BreakBlocks, terrain that hurts both player and enemy are Environmental Hazards, etc.)



* Scrum Backlog
* Object interactions began being worked on in early October after several objects, both weapons and enemies, had already had their base behavior created (*not* finalized)
* Anything created to deal with interactions whether it be object tags or scripts are continuously refined over time as more situations are prepared and unique behavior scenarios have to be accounted for
* Coding
* C# Scripts
* Numerous unique scripts exist for more complex object behaviors (The UFO, enemies, the guided missile), but several generic scripts also exist to handle simple behaviors that can be applied to multiple objects (a script that moves a projectile in a set direction at a set speed, a script that tells how much damage an object will do to what it's hurting, a script that says the attached object is an explosive and detonates into a unique explosion)
* User training
* Basic controls and a description of the gameplay loop is included in the manual (aka the README), but the user will have to observe how objects behave and discover how everything acts in relation to each other through inference and experimentation themselves (it's just more fun that way)
* Testing
* Interactions are typically tested and noted as a new object's base behavior is getting made, although in the early stages it wasn't a priority
* Dedicated testing to a new object didn't begin until the base behavior has reached a satisfactory state (no point in preparing for every scenario if you aren't sure about *everything* it can do)
* Dedicated testing follows a pattern of reusing preexisting scripts or creating new ones and **very** frequent debugging sessions. During this time, older objects are getting tested too through interactions with whatever's new.

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

* Final software/hardware product
* One fully fleshed out level with 9 items, and 4 enemy types.
* Original vision fully achieved, with big plans for the future.
* Source code and user manual – screenshots as needed - Technical report
* <https://github.com/jgschuma/UFO-Game>
* <https://chimmite.itch.io/ufo-game>
* User manual and screenshots included in the download
* Evaluation by client and instructor
* Team Member Descriptions
* Austin Matias
* Sounds
* Enemy Design
* Scoring Design
* Project Management
* Jacob Schumacher
* Item Management
* Item Powers
* Item Handling
* UFO Beam Design
* Copper Hicks
* Art/Animation
* Level Design
* Player Movement/Physics
* Item Dropoff Design