Ribbit Royale

Team Info

Aidan Caughey:	Team Lead / Art
Adam Bobich:	Project Manager / Gameplay Developer
Baron Baker:	Gameplay Dev / Art
Chase Bennett:	Sound Dev
Luke Garci:	UI / Multiplayer Setup
Ryan Dobkin:	Gameplay Systems Dev / Shaders

Communication:

We've set up a text group chat and discord server for general, quick communication. Additionally, we've set up a Trello board to track which tasks still need to be completed, which tasks each person is currently working on, and which tasks each person has already completed. We'll also use email as necessary.

Rules:

- Texts are expected to be seen and responded to (if necessary) within a 24 hour period
- If a task appears more difficult than you initially thought and you find yourself in over your head, you're expected to reach out for assistance as early as possible.
- Treat others how you would like to be treated, focus on constructive criticism rather than just being mean.
- Keep your mind open on ideas. No one concept is concrete and everyone's opinions should be taken into account.

  Github Repo for Project: https://github.com/abobich675/Ribbit-Royale
  Trello link: https://trello.com/b/ebDvr63Y/software-engineering-2

Product Description

Title: Ribbit Royale

Abstract: Ribbit Royale is an exciting and whimsical LAN party game designed to bring players together through fun, competitive, and engaging multiplayer minigames. Featuring a playful art style and a variety of unique, frog-themed interactions, the game invites players to experience amusing challenges that leverage the distinct abilities of frogs—such as swinging with their tongues, licking to interact with the environment, and even eating other players or flies. The objective is to provide a distinctive experience that blends competitive gameplay with quirky mechanics, making each mini-game feel fresh and enjoyable. Players can compete in up to four-player multiplayer mode, with a series of mini-games designed to test their skills and cooperation. Ribbit Royale offers at least three unique games with visually differentiated elements that make each challenge memorable and fun. Through the creation of this game, we aim to push the boundaries of the typical party game genre by incorporating fun mechanics that highlight the playful essence of frogs, making it not only an entertaining experience for players but also an intriguing marketable product. As a team of newcomers to game development, this project also provides a unique opportunity to grow our skills in game design, networking, and the Unity engine, while contributing to a project that could offer both personal and professional rewards.

Goal: Create a functional multiplayer party game that supports up to 4 players. Create at least 3 functioning party games with key visual aspects that differentiate each game from the next.

Current Practice: Most multiplayer party games, such as *Mario Party*, *Gang Beasts*, and *Fall Guys*, utilize engines like Unity and Unreal Engine for game development. These games focus on fast-paced, competitive, or cooperative minigames, often relying on traditional movement mechanics like running, jumping, and grabbing. Many of these titles emphasize accessibility and simple controls to appeal to a broad audience. However, many of these games cost upwards of $30 dollars and games like *Mario Party* even go the extra mile by only being playable on the Nintendo Switch, which costs another $200-300.

Novelty: Unlike typical party games, Ribbit Royale intends to break the norm by releasing entirely free to play, making it much more accessible than the games mentioned above. Additionally, Ribbit Royale differentiates itself through its frog-based movement and interaction mechanics. Instead of basic jumping or grabbing, players can:

- **Swing using their tongues** to traverse obstacles dynamically.
- **Lick to interact with objects and complete minigame tasks** in unconventional ways.
- **Hop like a frog** to race to the finish line
- **Ribbit**

These mechanics add playful chaos and strategic depth to the gameplay, making Ribbit Royale stand out from other party games that rely on traditional movement and interactions. The combination of unique physics-driven mechanics and engaging multiplayer competition sets our game apart in the party game genre.

Effects: Create a video game that puts a fun and unique twist on an existing popular genre, giving us the opportunity to create a game that is not only fun and entertaining to us, but potentially has market possibilities. Developing this game would also give all of us experience in a field we haven't worked in in-depth before.

Technical Approach: The development of Ribbit Royale will be done using Unity, a powerful game engine that will provide us with an extensive range of tools for game development, multiplayer networking and physics simulations. Unity's versatility allows us to create dynamic, interactive environments, and with its robust support for both LAN and online multiplayer, it is an ideal platform for our project. To streamline our workflow and ensure efficient collaboration, we will leverage Unity's cloud-based collaboration system, which allows for version control and seamless data sharing among the team. This will enable multiple developers to work on different aspects of the project concurrently without overwriting each other's progress.

Risks: One of the risks of our project is that multiple of our team members have never used the Unity software before, and have not worked on game development before. Because of this, we anticipate a learning curve that will slow down our project design. To mitigate this we will divide the work based on each person's strengths and weaknesses. Additionally, we'll prioritize communication with one another, helping if anyone falls behind and ensuring that each member learns the skills that they need to contribute.

Features

Main

Pre-game Lobby
LAN Multiplayer
3+ minigames

- Count the animal
- Rope swinging with tongues
- Parkour minigame, based on escape the dragon (Snake Escape)

Score Tracker/Leaderboard that will reward 1st, 2nd and 3rd place.

Stretches

Frog Character Customizer

Online Multiplayer

Use Cases (Functional Requirements)

Create a lobby to begin the matchmaking process - Luke

Actors	User wants to create a game lobby
Triggers	User clicks ‘create public’ or ‘create private’ button in the network lobby scene
Preconditions	User has launched the game
Postconditions (success scenario)	● User is in the network lobby scene waiting for others to join with a lobby code on screen ● Users are able to share a lobby code to friends.
List of steps (success scenario)	1. User launches the game 2. User clicks play game button 3. User clicks create game button 4. User enters a name for the lobby 5. User clicks create private or create public lobby 6. If user pressed public then display the lobby in the network lobby scene 7. If the user pressed private then the lobby will not be displayed in the network lobby scene
Extensions/variations of the success scenario	● User creates a public lobby ● User creates a private lobby
Exceptions: failure conditions and scenarios	● User has no internet connection and won't be able to connect to the server

Join any existing public lobby - Luke

Actors	User wants to join a random existing game
Triggers	User clicks ‘Join game’ button in the network lobby scene
Preconditions	• User has launched the game • There is a live public lobby • User has access to the internet
Postconditions (success scenario)	User is in the network lobby scene with the host and any other users who have joined
List of steps (success scenario)	1. User launches the game 2. User clicks play game button 3. User clicks the play game button
Extensions/variations of the success scenario	•
Exceptions: failure conditions and scenarios	• User has no internet connection and won't be able to connect to the server • There are no lobbies that are available to join

Join a lobby using the lobby code - Luke

Actors	User wants to join a lobby using the lobby code
Triggers	User clicks ‘Join code’ button in the network lobby scene after putting in a valid lobby code
Preconditions	• User has launched the game • There is a lobby (private or public) with a lobby code
Postconditions (success scenario)	User is in the network lobby scene with the host and any other users who have joined

List of steps (success scenario)	1. User launches the game 2. User clicks play game button 3. User clicks the play game button
Extensions/variations of the success scenario	●
Exceptions: failure conditions and scenarios	● User has no internet connection and won't be able to connect to the server

Change the color of the frog sprite- Luke

Actors	User wants to change the color of the frog sprite
Triggers	User clicks one of the four color buttons
Preconditions	● User has launched the game ● User has joined or created a lobby
Postconditions (success scenario)	User is in the network lobby scene with the host and any other users who have joined
List of steps (success scenario)	1. User launches the game 2. User either creates or joins a lobby 3. User clicks one of the colored buttons
Extensions/variations of the success scenario	● The frog sprite's color updates to the selected color. ● Other players in the lobby can see the updated color
Exceptions: failure conditions and scenarios	● Another player in the lobby has already chosen that color

Ready up in the network lobby scene to start the game- Luke

Actors	User wants to start the game

Triggers	User clicks the ready up button
Preconditions	• User has launched the game • User has joined or created a lobby • User has a color selected
Postconditions (success scenario)	User is taken to the pre lobby where they are able to pick the minigame they want to play
List of steps (success scenario)	1. User launches the game 2. User either creates or joins a lobby 3. User has chosen the color they want 4. User clicks the ready up button 5. If all players in the lobby are ready, the game transitions to the pre-lobby. 6. Users can now select a minigame.
Extensions/variations of the success scenario	•
Exceptions: failure conditions and scenarios	• Not all players in the lobby have pressed ready up

Check the Scoreboard - Ryan

Actors	User wants to check their score
Triggers	User presses the scoreboard trigger button, e.g. 'Tab'
Preconditions	The player is currently in a game and does not already have the scoreboard pulled up.
Postconditions (success scenario)	User successfully opens the scoreboard overlay, views their score, then closes the overlay.
List of steps (success scenario)	The user launches the game. The user then joins a game or creates a new game. Once the players start scoring, the user checks the scoreboard to see their score, and their rank on

	the scoreboard.
Extensions/variations of the success scenario	The player views the scoreboard as players score, in which case the ranks will animate a change in position. The player views the leaderboard before the game begins, in which case the scores will be 0. The player also wants to view rank or their team color, in which case that will also appear when they open the scoreboard.
Exceptions: failure conditions and scenarios	If the game is not started, the scoreboard will not be available and the user will fail to see their score. If the user does not have the scoreboard button bound, they will have to wait until the end of the round to see their score, wherein it will automatically appear. If the player is unsure of their team color, they may be unable to identify which score is theirs.

Tongue Swing Game - Aidan

The players will compete against each other to swing and race to get to the top platform first.

Actors	User controls their frog character to swing their tongue to reach a target platform
Triggers	Player is prompted to use their tongue to swing like a vine
Preconditions	• The player is in the tongue-swinging minigame • Swingable objects are present and within range of the player’s tongue • The player has control of the frog character
Postconditions (success scenario)	• The player successfully swings • The game updates the player’s progress
List of steps (success	1. Player presses an action button that will

scenario)	extend the frog's tongue 2. The tongue latches onto a nearby swingable object 3. The frog swings back and forth 4. The player releases their tongue at the right moment and goes flying
Extensions/variations of the success scenario	• Whoever gets to the target platform first wins the minigame
Exceptions: failure conditions and scenarios	• The player releases the tongue too early or too late resulting in a failure to launch themselves far • Another player disrupts another player's swing by colliding or ribbitting

Options Menu - Aidan

Users will be able to change options in the Main Menu screen that allow for display and sound options.

Actors	User wants to change their settings in the options menu
Triggers	Player clicks the options button on the main menu
Preconditions	• The player is in the main menu screen • The player clicks the options button
Postconditions (success scenario)	• The player has successfully changed his options
List of steps (success scenario)	1. Player presses the options button 2. The options menu pops up 3. Player changes their settings 4. Player presses the back button 5. Options menu closes
Extensions/variations of the success scenario	• Player does not change their settings but still is able to successfully go back
Exceptions: failure conditions and scenarios	n/a

Snake Escape - Baron

Actors	Users control their frog character to jump across gaps or obstacles to escape a snake.
Triggers	The game begins when the players enter the gameplay area, initiating the snake chase.
Preconditions	● The level environment with gaps, obstacles, and the pursuing snake is generated. ● The game input is functioning and responsive.
Postconditions (success scenario)	● The frog successfully escapes the snake by reaching a safe zone or surviving for a set duration. ● The game rewards the player with points, a higher score, or not losing a life.
List of steps (success scenario)	1. The snake begins chasing the frog characters as the level starts. 2. The player uses controls to make the frog jump over gaps or obstacles. 3. The environment dynamically changes as the frog progresses. 4. The player times jumps correctly to avoid falling or being caught. 5. The player successfully reaches the safe zone or survives for the required time, ending the chase sequence.
Extensions/variations of the success scenario	● The player collects bonus items, such as flies or power-ups, while evading the snake. ● The frog encounters different types of obstacles, such as logs, moving platforms, or slippery surfaces.
Exceptions: failure conditions and scenarios	● The player misses a jump, and the frog falls, resulting in game over. ● Snake catches the frog: The player

	delays movement or jumps too late, allowing the snake to catch the frog.

Count the Animals - Chase

Actors	User inputs the number of animals they have counted.
Triggers	The counting animal mini-game finishes, and a screen pops up prompting for an answer. The correct answer is stored to be compared to.
Preconditions	In the counting animal mini-game
Postconditions (success scenario)	• User inputs the correct amount of animals. • User player's progress gets updated.
List of steps (success scenario)	1. Animal counting game ends 2. Prompts users for answer 3. User enters answers 4. Player Closest to the answer wins 5. Winner gets +1 trophy 6. Return to main board
Extensions/variations of the success scenario	• If there are multiple winners (i.e the answer is 7 and 3 people guess 6.) They all get a trophy.
Exceptions: failure conditions and scenarios	• User gets it wrong, they get no points. • All users get it wrong, no one gets points.

Falling Rocks - Adam

Actors	User wants to start the game

Triggers	User selects a game
Preconditions	● All other players have voted for a game ● Players are in the prelobby ● Players are connected to a network
Postconditions (success scenario)	● A game is selected based on which game had the most votes in the prelobby ● User is taken to the the corresponding minigame, where the instructions are first displayed and then the game starts
List of steps (success scenario)	1. User launches the game 2. User either creates or joins a lobby 3. All users ready up and are taken to the prelobby 4. Each player moves their character using WASD to stand on top of whichever game they're voting for 5. When each player has voted, the game with the most votes is selected 6. All players are taken to the selected game
Extensions/variations of the success scenario	● When there is a tie for the game with the most votes, a game will be chosen at random out of those which are tied.
Exceptions: failure conditions and scenarios	● Not all players in the lobby chosen a game

Non-functional Requirements

1. The game must handle up to four players in a LAN setup with minimal latency.
2. The menu screen and user interface should be intuitive, with clear navigation and controls for players of all skill levels.
3. The game must prevent unauthorized access to the LAN session, ensuring only invited players can join, or require a password to enter the LAN session.

External Requirements

- The game should provide fallback mechanisms for errors, for example, providing a reconnect option if a player has disconnected.
- Provide a downloadable installer with clear instructions
- Include detailed build instructions in the Github repository to allow external developers to compile and run the game.

Team process description

i. Risk Assessment

Risk	**Likelihood**	**Impact**	**Evidence**	**Mitigation**
Lack of Unity Experience	High	High	A majority of the team hasn't worked with Unity before this class.	Unity tutorials, team training and helping each other out.
Networking Issues	Medium	High	If our game runs on high latency.	Early prototyping (which we've done), use of already established Unity networking tools and code.
Poor time management	Medium	High	We are all busy Computer Science students with 3 other CS classes,	Weekly check-ins with team and TA, Trello for task management and tracking,

			meaning a lot of work and we have to balance our time well between classes.	making sure everyone is on track.
Gameplay balance problems	High	Medium	Game balancing and tweaking values to make the gameplay perfect.	Iterative playtesting, external feedback.
Scope creep	Low	High	We had decided on three minigames, but if our time management takes the best of us, then our scope may need to be lowered.	Focus on core minigames first, then after add more features.

ii. Project Schedule

Date	Measurable	Effort
Jan 24	Setup Environment and File Sharing.	1 week
Jan 28	Implement basic controls including frog movements: left, right, up, down and jump.	0.5 weeks
Jan 31	Make the pre lobby map/area.	0.5 weeks
Feb 4	Create minigame 1 Prototype.	1 week
Feb 10	Create minigame 2 Prototype.	1 week

Feb 16	Create minigame 3 Prototype.	1 week
Feb 20	Prototype Playtesting with External Volunteers (Requires minigames 1-3 to have a working prototype)	0.5 weeks
Feb 21	Implement LAN Multiplayer (requires at least 1 game to have a working prototype)	1 week
Feb 24	(Stretch/Optional) Implement Online Multiplayer (requires LAN Multiplayer to be working)	1 week
Feb 22	Finish creating all art	1 week
Feb 24	Implement all art (requires all art to be finished)	1 week
Feb 28	Product Playtesting with LAN or Online Multiplayer (requires LAN to be working) (requires all 3 minigames to be completed)	0.5 weeks
March 2	Refine minigames based on testing feedback (requires product testing to be completed)	0.5 weeks

iii. Team Structure:

Team Roles:

- **Aidan Caughey (Team Lead/Art):** Responsible for overall project direction and the art direction and I'll create the assets / sprite work. Additionally, lead of testing, qa and problem detection.
- **Adam Bobich (Project Manager/Game Developer):** Responsible for keeping track of and assigning tasks to team members. Lead on Lickity Split (Tongue Swinging) and Swamp Spotting (Count the Animal) Games.
- **Baron Baker(Gameplay Dev / Art):** Responsible for the development of Snake Chase minigame. Also, I will help with asset / sprite work on level and character design. Additionally, I will be in charge of setting up team meetings with TAs.
- **Chase Bennett (Gameplay Dev / Sound Dev):** Responsible for sound design and sound scripts.

- **Luke Garci (UI/Multiplayer Setup):** Responsible for configuring the multiplayer aspect of the game. Did the networking of the entire project along with the user interface that allows a user to configure settings and start matches.
- **Ryan Dobkin (Gameplay Systems Dev):** Responsible for gameplay development, specifically regarding systems, such as character interaction, scorekeeping, leaderboards, etc.

Toolset:

- Unity for the game's development which provides a lot of tools for game development and even includes multiplayer functionality
- Trello for task management and tracking progress
- Github for version control and collaboration making sure everyone is working with the same version

Feedback:

Feedback would be most helpful after our first minigame has been implemented. Playtests with non-team members to refine gameplay and identify usability issues from there.

iv. Test Plan & Bugs

Planned Tests:

- **Unit Testing (Input Handling, Physics, Game Logic)**
- **Network Testing (LAN, Latency handling, Synchronization)**
- **UI Testing (Scoreboard, HUD)**
- **Performance Testing (Max Players, Network Load, FPS stability)**
- **External Playtesting**

Test-Automation Infrastructure:

We are using NUnit for unit and integration testing, which is the standard for testing in Unity. Additionally, we will use Unity's built-in Play Mode and Edit Mode test frameworks to verify in-game behavior.

Adding a New Test:
To add a new test, navigate to the Assets/Tests/ directory in the Unity project. From there, choose either to create an EditMode or PlayMode test depending on what you're testing. Go into your selected directory and create a new test script (NewTest.cs) and use NUnit.Framework for assertions. Once you commit your script, CI will test your script and any others in the directory.

Continuous Integration Service:

CI Service	Pros	Cons
GitHub Actions	Integrated with GitHub, free and flexible workflows	Limited to GitHub repositories
CircleCI	Fast build times and good for docker projects	Complex setup and configuration
Jenkins	Highly customizable and supports many platforms	Complex setup and maintenance

We are using **GitHub Actions** to automate testing and build processes as it's the easiest to implement and can use Unity frameworks in workflows.

CI Tests:

- **Unit Tests (Edit Mode):** Scoreboard and Multiplayer tests to ensure consistency.
- **Validation Tests (Play Mode):** Color ID tests to ensure correct colors.
- **Integration tests (Play Mode):** Testing games in full environment.
- **System tests (Play Mode):** Loading each individual minigame to see if all assets load and build correctly.

What Triggers a CI Build:

- On push to master to verify commits don't break the build
- On pull request to ensure new code is tested before merging.

v. Documentation Plan

- **User Guide:** Game installation, controls and gameplay mechanics.
- **Developer Guide:** how we structured code, implemented minigames and set up networks.

- **In-Game Help:** Instead of relying on external tools to teach player's how to play the game, we want to make sure everything is easy to understand without having to exit the game. Therefore, we want to place a high priority on having each button, icon, or game mechanic make intuitive sense. For elements that cannot be conveyed so intuitively, such as game controls or objectives, we will include tutorial screens to explain prior to the game starting.

Software Architecture

Components:

- **Main Menu**
  - Allows the user to choose between joining an existing multiplayer session or joining a new one.
- **Network Lobby**
  - User is able to create a game which will allow other players to join their lobby. Players can press join game and be put into an existing lobby
- **Character Select**
  - Users are able to choose the color of their frog, ready up or go back to the main menu. If all users ready up then they will be moved into the pre lobby.
- **Pre-Lobby**
  - All users can move around freely using W,A,S,D and after a certain amount of time the users will be put into a minigame. Users are sent back to the Pre-Lobby after each minigame to view the leaderboard.
- **Minigames**
  - Minigame modules each have unique player controllers, physics and game logic. Players are sent into minigames from the pre-lobby.
- **Player Controller**
  - Allows for user interaction and input handling.
- **Score Manager**
  - Manages every player's scores and displays it with the Scoreboard UI.

Event Flow:

- **Lobby Phase**
- **Minigame Selection**
- **Gameplay Interaction:** Player Controller triggers -> System processes result -> player score updates.

- **Game End:** Minigame completed -> Pre-Lobby -> Repeat until GameEnded is fired.

Interfaces:

- **Main Menu <-> Network Lobby**: Synchronizes the game state across clients.
- **Pre-Lobby <-> Minigames:** Controls game transitions between minigames and the logic execution of each minigame.
- **Player Controller <-> Minigames:** Enables player interactions within minigames.
- **Score Manager <-> Network Lobby:** Score is updated for each player.

Data:

- **Session Data:** Stored in memory, synchronized across clients via Unity's networking. Includes things like sessionID and clientID.
- **Game State:** Managed by the Game Manager. Includes things like player positions (+ velocities and other physics), game timers and game completion (per player).
- **Player Scores:** Tracked locally and updated on all clients. Tallied across multiple games.

Architectural Assumptions:

- Four players will be supported
- The game will run on LAN with low latency
- Players will have stable connections without the need for server-side connections.
- Unity's event system ensures decoupling and scalability.
- LAN-based low-latency communication synchronizes event-driven actions.
- Reliable event handling mechanisms help mitigate synchronization issues.

Alternative Architecture:

- **Cloud-based Multiplayer:** Stretch goal we have is online multiplayer. Enables global play with persistent lobbies, but will have higher latency and would require an external server setup.

- **Peer-to-Peer Networking:** Removes the reliance on a dedicated host, but would have more complex synchronization and also could cause potential security issues.

Software Design

Component Breakdown:

Component	Scripts	Pseudocode
Main Menu	**MainMenuUI.cs**	User clicks Play button and sends him to the Network Lobby User clicks quit and will close the game window
	Loader.cs	Handles loading scenes, both for single-player and multiplayer games using Unity Netcode
Network Lobby	**LobbyUI.cs**	User clicks create game which creates a host instance User clicks the join game button which creates a client instance
	RibbitRoyaleMultiplayer.cs	Handles multiplayer functionality and synchronizes player information
Character Select	**CharacterSelectionUI.cs**	Select your frog color User clicks ready button which will then show they're ready above their head
	CharacterSelectReady	If all clients are ready, send

		to Pre-Lobby scene
Pre-Lobby	**PreLobbyManager.cs**	Once players connect, they appear on the pre-lobby scene This controls game state (pre-lobby, minigames, end screen) Either connects clients or/and loads scenes and spawns players in
Minigames	**Lickity Split (Tongue Swinging)**	A/D controls the player's horizontal movement. The mouse controls the player character's tongue (If the mouse is clicked, create a tongue object and attach to the nearest node if in range. If the mouse is released, destroy the tongue connection). Uses the Unity Spring Joint to attach the frog to the node. When swinging, detect which direction the character is swinging and give a little bonus movement in that direction for more satisfying swings. Detect when the player reaches the end platform, save their time, and compare it to the other player's times to decide the winner.
	Swamp Spotting (Count the Animal)	Count the specific animal

		Various animals walk across screen for 30 seconds Question prompting how many of a specific animal Players press interact button to increment their count Score is given based on how close players are to the actual number
	Snake Escape	The objective is to run away from the snake until you reach the finish line or lose the minigame. Players can control the frog to move left, right, and up. Players will have to jump from platform to platform, across trees or puddles, etc. Points are awarded depending on placement among players.
Score Manager	**ScoreManager**	Score Manager creates and keeps track of scoreboards and player scores. It does this by creating an entry for each player/color chosen through Character Select. This is overlaid as a GUI, which is toggleable. If a token is collected, the player's score is updated. All player's rank is updated upon a change in score.

	TokenInstance	Gives functionality to token-type objects. When a player collides with a token instance object, the object will act as a token. The correct player score will be incremented, the token will remove collision and do a bounce, then disappear.

Coding Guidelines

Coding Style Guidelines:

- C# (Unity)
- Shader (HLSL)

Enforcement:

- These guidelines align with industry standards and best practices
- Code reviews will ensure compliance of these guidelines

Reflections:

Aidan Caughey:

- Having a more comprehensive and deeper planning phase to get all ideas fleshed out and what we want to implement set in stone before the actual implementation phase. Having this deeper planning phase will allow for less problems like scope creep happening which can seriously affect the implementation phase and also team morale.
- Making sure to test more oftenly, basically whenever a new feature is implemented is important to me. Automating these tests ensures it will be correct every time.
- I liked that our team was flexible and communicative and we all were able to focus on different aspects of the project.

Adam Bobich:

- Each teammate is going to have their own strengths and weaknesses, as well as concepts that they're more knowledgeable on are more efficient at implementing. Therefore, it's very important to designate each member of

the team with specific sections of the project (and even specific individual tasks).
- Similarly, it’s important to communicate with your team members and ask for help when it comes time to implement a feature that’s similar to (or is highly involved with) a feature already implemented by a different team member. This way, less time is wasted relearning ideas from scratch when an expert already exists on the team.
- Finally, it’s important to test well and test often. This of course means automating test cases, but also testing the project after adding each new feature and ideally building and testing the project before each and every push to the master github branch. This way, there’s no issues of looking through past commits on the master branch to try to pinpoint where the issue popped up, since you’ll catch it right when it happens.

Baron Baker:
- I learned that seeking help from teammates when implementing features that are similar to one’s they have already worked on to be helpful. Less time is wasted this way, and I can implement it in a similar fashion to my teammates so the readability stays the same.
- Finding a form of communication that we all use on a daily basis was a great way for us to stay up to date on what each other we’re doing in the project. This also allowed us to work on different aspects of the project separately, but together at the same time.
- Having stricter deadlines on when certain parts were implemented would have helped with a smoother development process near the end of the development cycle.

Chase Bennett:
- Having a more in depth planning phase like Aidan said, would’ve helped a lot so everyone can get their ideas out there and we could really decide on what ideas we want to continue with and what ideas we don’t do.
- Communication was great and a lot of us were flexible which was great
- A more in-depth development plan and soft deadlines to help ourselves guide us more along the way.

Luke Garci:
- Have stricter deadlines for certain implementations of our project for a smoother development experience
- Spend more time planning and researching how our plans will work with the tools we used to build our project and if they are the best choices
- Team communication between members working on different parts of the project to help integrate all aspects of the project was good

Ryan Dobkin:

- Projects can get disorganized and hard to keep track of fast, keeping track of goals and deadlines should be a priority. Specifically, progress reports or (bi)weekly meetings would've helped us coordinate better and would have helped us from having to pull all nighters at the very end to get things done
- Without proper communication and or planning, 'code smells' such as duplicate code and uncommented functions can pile up very quickly – planning a more definitive project outline, then revising the outline should requirements change, should be the top priority I learned
- Refactoring should be a more integral part of the coding process than I think it was for us. As our codebase grew, so did our interdependence and redundancies. With me specifically, I had several instances where I revised program logic several times, having to rewrite functions and classes each time. These piled up and often were not any simpler or well structured than the previous iterations, causing a deepening code debt. This was also partly caused by me not knowing how to use Unity for the first few weeks, but continuing to use my bad code through the end of the project instead of refactoring. – I learned the importance of refactoring; that refactoring, especially for multi-person and multi-month projects like ours, is not optional, and should've been a core part of our development plan.