Project Proposal

1. Team Info

Aidan Caughey: Team Lead / Art
- Email: caugheya@oregonstate.edu | 503-314-8224

Adam Bobich: Project Manager / Unity Expert
- Email: bobicha@oregonstate.edu | 650-346-2153

Baron Baker: Gameplay Dev / Art
- Email: bakerbar@oregonstate.edu | 310 - 692 - 5420

Chase Bennett: Gameplay Dev / Sound Dev
- Email: bennchas@oregonstate.edu | 541-829-0572

Luke Garci: UI / Multiplayer Setup
- Email: garcil@oregonstate.edu | 480-244-1670

Ryan Dobkin: Gameplay Systems Dev / Shaders
- Email: dobkinr@oregonstate.edu | 949-922-4482

Github Repo for Living Document: https://github.com/abobich675/SoftwareEngineering2
Github Repo for Unity Project: https://github.com/abobich675/Ribbit-Royale
Trello link: https://trello.com/b/ebDvr63Y/software-engineering-2
Thursday 2:45-3:00 TA Meeting BEXELL 322

Communication:

We've set up a text group chat for general, quick communication. Additionally, we've set up a Trello board to communicate 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.

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: Current multiplayer games are built using game engines like Unity or Unreal Engine that provide tools for 3D/2D game development, networking, and physics.

Novelty: As a game themed and designed around playful and unique frog interactions, we can incorporate novel features like swinging with your tongue, licking to interact with tasks, or even eating other players or flies. This would differentiate our game from existing party games as we would have games with unique interaction means and objectives, allowing us to stand out.

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-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.

3. Features

Main

Home Screen
LAN Multiplayer
3+ minigames

- Count the animal
- Rope swinging with tongues
- Parkour minigame, based on escape the dragon (Snake Escape)
- Falling tile game
- Frogger minigame, players can have some form of interaction with one another

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

Stretches

Frog Character Customizer

Online Multiplayer

4. Use Cases (Functional Requirements)

Use Case 1- Luke

Actors	User wants to play against their friends
Triggers	User clicks ‘create lobby’ button in the main menu
Preconditions	User has launched the game and has pressed play game.
Postconditions (success scenario)	User is in a pre-lobby waiting for others to join with a lobby code on screen. User gives a lobby code to friends, once there are 4 players, a countdown begins before starting the party games.
List of steps (success scenario)	The user will launch the game. The user will then press play. The user will press create lobby. The user will be able to show/tell the lobby code to their friends so that they are able to play.
Extensions/variations of the success scenario	The user can manually start the game in the lobby with less than the recommended amount of players. All 4 players need to ready up to begin playing.
Exceptions: failure conditions and scenarios	If the game is not started and there is less than 4 players when the timer is finished, then the lobby is destroyed.

Use Case 2- Ryan

Actors	Users looking to improve at a specific minigame

Triggers	Users open the game and select the minigame they wish to practice, then select play
Preconditions	The user has a desire to improve their skills or ability to complete the objectives required for the minigame to beat their friends
Postconditions (success scenario)	The user finishes the minigame in first place, satisfied with their improved skills.
List of steps (success scenario)	The user opens the game, selects a minigame they wish to improve in, and selects play. The user plays the minigame and attempts to finish in first place. The user finishes in first place, and is now satisfied with their improved skills.
Extensions/variations of the success scenario	The user opens the game, selects a minigame they wish to improve in, then selects play. The user plays the minigame and attempts to finish in first place. Before they are able to place first, they have to leave the room. As this game is multiplayer, they are unable to pause, so will have to restart their attempt at getting first. The user may also choose a different minigame to improve upon before receiving first in the starting minigame, allowing the user to be satisfied but not necessarily fulfilling the first minigame's success condition.
Exceptions: failure conditions and scenarios	The user opens the game and selects a minigame they wish to improve in, then selects play. The user plays the minigame and attempts to finish in first place. The user is unable to finish in first place after multiple tries. The user exits the game dissatisfied. The user may also be forced to quit, either through external intervention or game malfunction, forcing them to be unable to complete the task.

Use Case 3- Aidan

Actors	User controls their frog character to swing as far as possible with their tongue
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 and lands • The game updates the player's progress
List of steps (success scenario)	1. Player presses an action button that will 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 5. Player lands and his distance is recorded
Extensions/variations of the success scenario	• After every player goes, they tally up how far each went and whoever went the farthest 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 ribbitting

Use Case 4- 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.

Use Case 5- 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.

Use Case 6- Adam

Actors	User controls their frog character, making them run/jump from rock to rock to avoid falling into the water
Triggers	Player is standing on a stationary rock
Preconditions	• The player is in the Sinking Rocks minigame • Player is standing on a rock • The player has control of the frog

	character
Postconditions (success scenario)	• The rock sinks and subsequently disappears • The player's frog is no longer on the rock
List of steps (success scenario)	1. Player moves their frog onto the rock 2. The rock starts to shake 3. After a delay, the rock starts to sink 4. The player moves the frog to a neighboring rock. 5. The first rock completely sinks and disappears 6. The frog is now standing on a new rock. Repeat from 2
Extensions/variations of the success scenario	• Players can make their frog jump across a small gap to get from one rock to another • All other players get eliminated and the rock stops shaking/sinking. The player wins and the everyone moves on to the next game
Exceptions: failure conditions and scenarios	• If the player fails to move their frog onto another rock, or there are no adjacent rocks to jump to, the frog will sink with the rock and swim away, losing the minigame • The player walks off the edge, sinks, loses, and swims away.

5. 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.

6. 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.

7. 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, meaning a lot of work and we have to balance our time well	Weekly check-ins with team and TA, Trello for task management and tracking, making sure everyone is on track.

			between classes.	
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 24th	Setup Environment and File Sharing.	1 week
Jan 28th	Implement basic controls including frog movements: left, right, up, down and jump.	1 week
Jan 31th	Make the pre lobby map/area.	1 week
Feb 4th	Create minigame 1 Prototype.	1 week
Feb 9th	Create minigame 2 Prototype.	2 weeks
Feb 14th	Create minigame 3 Prototype.	2 weeks
Feb 19th	Implement LAN Multiplayer	1 week

Feb 24th	Refine minigames	2 weeks
March 3rd	Final testing	2 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.
- **Adam Bobich (Project Manager/Unity Expert):** Responsible for keeping track of and assigning tasks to team members. In charge of teaching members who don't know as much about Unity and how to implement their ideas.
- **Baron Baker(Gameplay Dev / Art):** Responsible for the development of overall gameplay like character functionality, minigames, and lobby play. 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 gameplay development such as minigames, main board, home screen. Also will handle the sound effects, and getting all the sound lines in order.
- **Luke Garci (UI/Multiplayer Setup):** Responsible for configuring the multiplayer aspect of the game. Along with the user interface that allows a user to configure settings and start matches.
- **Ryan Dobkin (Gameplay Systems Dev / Shaders):** Responsible for gameplay development, specifically regarding systems, such as character interaction, scorekeeping, leaderboards, etc. Experienced in writing and working with OpenGL Shaders (GLSL) and Unity Shaders (HLSL), will apply knowledge to create and help with shader creation.

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

- **Component Testing:** Each component will have dedicated test cases.
- **Integration Testing:** Multiplayer synchronization and minigame interactions.
- **Usability Testing:** Playtesting with external users for feedback.
- **Bug Tracking:** Managed via Github Issues.

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:** UI tips and tutorial elements.

8. Software Architecture

Components:

- **Game Manager:** Handles the overall game state, minigame transitions and leaderboard scoring.
- **Network Manager:** Manages player connections, lobby setup, client and host and the synchronization of the game state.
- **Player Controller:** Handles movement, actions and interactions within minigames and in the lobby.
- **Minigame Modules:** Self-contained logic for each minigame, including physics and win conditions.
- **UI System:** Provides a consistent interface for the main menu, lobby, settings and in-Game GUI.

Interfaces:

- **Game Manager <-> Network Manager**: Synchronizes the game state across clients.

- **Game Manager <-> Minigame Modules:** Controls game transitions between minigames and the logic execution of each minigame.
- **Player Controller <-> Minigame Modules:** Enables player interactions within minigames.
- **UI System <-> Game Manager:** Provides players with game status and options.

Data:

- **Session Data:** Stored in memory, synchronized across clients via Unity's networking.
- **Game State:** Managed by the Game Manager and synchronized via the Network Manager.
- **Player Scores:** Tracked locally and updated on all clients.

We have no need to have an external database, as all data exists within active sessions of the game with no need for persistent storage. Only thing we could store would be the number of wins you have all-time, which can be done locally.

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.

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.

9. Software Design

Component Breakdown:

- **Game Manager**
    - Controls game states and transitions

    - Tracks player progress and initiates mini game logic
    - Unity Game Engine
- **Network Manager**
    - Handles player connections
    - Ensures consistent game state
    - Unity's networking API
- **Player Controller**
    - Manages the movement and interactions
    - Syncs with the Network Manager for consistency across players
- **Minigame Modules**
    - Each minigame has its own logic for physics, scoring and win conditions
    - Implements abstraction for modularity
- **UI System**
    - Provides menu navigation, game HUD and endgame results
    - Unity's Canvas system for rendering

10. 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