**Lunar Landers Final Report Summary**

**Group 12: Jacob DiStefano, Dan Hrubec, Julian Gonzales, John Mistica**

**Project Description**

Lunar Landers is a multiplayer extension of the classic arcade game, Lunar Lander. The game pits multiple players against each other to see who is able to land their spacecraft on the surface first and prevent their opponents from doing so. New challenges may face the player during their match may include lunar wind and worms, in addition to the threat of being blown away by their opponents.

This game exists in the casual subset of the games industry, which exists for players who do not take video games terribly seriously. The work addressed by this product is to give quick simple entertainment to casual video game players. It does this by challenging the player and pitting them against other players.

The work handled by the product would be to create an environment where players can pick up and play the game. As a result, the product would need to have a strong server established to allow client connections and create matches for players to play. Allowing for an entertaining yet simple UI to allow for players to appreciate the game and its simplicity.

The product must be an online multiplayer game. This will create an engaging experience that will keep users playing. It must also be compatible with mobile devices. This is because most of the population and younger population have a mobile device and can play the game and create a larger audience.

**Requirements**

Two of the essential use cases for players involve which sort of match they would like to queue for. If the players decide to join an unranked lobby, then it will just wait for enough players before creating a game.If a player chooses to enter into a ranked game lobby, the server will have to group people within a certain range based on skill so the game remains competitive, and that players are evenly matched.

The Functional Requirements for this project mostly relate to the creation of matches, the queuing of players, and the determination of victors. The maps generated for each match must be distinct and must be enjoyable to play on separate matches, and will be manually evaluated to ensure variety in map types. The player queue is the mechanism in which matches are made and players are placed into matches, this system must be able to properly place 8 players in each match. This system must be able to endure strain from a great amount of players queuing at once, and allow for party play. The winner and scoring calculations must be able to analyze a player’s performance and assign a score, and determine a victor. This system must be able to function in any possible match, and needs to be tested against all possible score-able actions.

The Performance Requirements ensure that the game functions accurately and precisely during normal gameplay. The game must allow for tolerable speed and latency, ensuring that a player’s input and the corresponding action occur in quick enough succession to effectively control the lander and react to opponent input. The collisions between players and objects (or other players) must be precise and consistent in order to allow for fair, balanced and enjoyable gameplay. The game must have the capacity to keep track of each player’s ship, and all obstacles on the map in order to effectively react to each player’s input. This must be done with minimal strain on both the server side host and the client’s machine.

All requirements, including those not listed above, have robust acceptance tests. These will be done upon on release, and upon each update to ensure that the game still meets all of the requirements it must meet to be acceptable to the clients and developers.

**Design**

The design goals focus on a simple and easy to use user interface to allow players to quickly create and join games. Another major design goal is to have a unique map generation via the use of a seeding system, so that no maps are identical to others, so each map is a unique and new experience.

The system architecture used for matches will be the classic Client Server architecture, with Client inputs sent to the server for processing and responses sent back to the player to update the game’s display. The server will keep track of statistics pertaining to player’s gameplay, and will update the client if the client wishes to see their statistics.

The system was initially decomposed into multiple subsystems; Client, Server, Match, Queue. The Client subsystem is responsible for the user interface as well as utilizing resources to send players into the queue. The server subsystem connects the players from the queue into an actual match. The server has an open port in which the players connect to when finding a match. The match subsystem is the actual game, which has a seed and generates a new map accordingly. Players all connect and interact with each other on this match and map. This subsystem also handles game logic such as physics and AI for opponents or stage hazards. The queue subsystem pertains to the server, and listens in on a port. Depending on the type of match being played, will either wait for enough players and send them into a game, or wait for enough players within the same skill rating to send them into a match.

**Project Issues**

We have yet to determine whether queuing and matchmaking should be split into ranked pools, or all players competing with one another in one large pool. Each of these methods have benefits and drawbacks, such as the splintering of the community for ranked pools and the dominance of veteran players in an unranked pool. The matchmaking algorithm must be deliberated further to determine the course of action.

The inclusion of multiple game modes, such as a classic mode with no hazards or weather conditions, has not been determined yet. Currently we believe there should only be one game mode, yet we would not want to force players to play in a way which is too confusing or not engaging for them.

We have considered several third party game engines to build our game in, rather than developing an in house engine. This would allow for faster development and more stability. We feel that Unity would be best suited for the development of this game. It would make implementation much easier, especially due to its included physics engine. Unity’s extensive toolset for the development of two dimensional games makes it a promising choice for development. In addition Unity is very easy to port between skews, and would allow for wider adoption of the game.