**City Based RISK Game Final Summary**

Group 4 - Luqmaan Baiyat, Joshua Herman, Hamza Shahid, Patrick Gundry

The project will be a version of the board game RISK. In this adaptation, there would be many different maps for the players to play on. Each map would be comprised of intersections of a district in a city that players choose. Each intersection must have at least two streets touching each other, but in theory could have no maximum limit of streets intersecting. Players would select which map they would like to play in, and the game would use the Google Maps API to download map data of the selected district in the selected city. From this map data, a virtual battlefield will be created over the district with the street intersections as the points of interest during a match.  
  
 Players in the game control different intersections with military units and can attack other players’ intersections to gain control of them. At the beginning of the game, the intersections are split evenly between the players, and they distribute a dynamic number of units across their controlled intersections. The number of units each player begins with is inversely proportional to the total number of players in the match. In order to fortify their controlled intersections players will be given additional units to place on their intersections over the course of the game. At the start of each turn, players will be given a static amount of units to distribute across their intersections. Additionally, players can use *intersection cards* to gain additional units to distribute. At the beginning of the game, the bonus number of troops for cashing in intersection cards will be a small amount, but will scale up as more players cash in intersection cards. An intersection card will be labeled as infantry, jeep, or tank. To get the card bonus, the player turns in three of any individual labeled card, or one of each labeled card. Additionally, the player can use wildcards to substitute any card they’d like to turn in for their unit bonus.  
  
 After placing their bonus units, players can choose to make attacks during their turn, move troops around to adjacent controlled intersections, or skip to the next player’s turn. Attacking or defending an intersection is based on how many units are on both intersections as well as random number rolls from both players. Players can only attack intersections that are adjacent to intersections that they control and requires the intersection the attack was initiated from to have at least two units. There is no limit to how many attacks a player can initiate in a single turn, provided they meet the rules to initiate an attack. A player loses control of an intersection if all their units there are killed. Possession of the intersection is transferred to the player who eliminated the last unit from that territory, and that player can now place their units to defend that territory. After a player captures an intersection from another player, they are given an intersection card. A player is the winner if they control all intersections in the district. If a player loses control of all their intersections, they are eliminated from the game.  
  
 The functional requirements for this project are a single-player mode, multiplayer mode, and tournament mode. Having these different modes for the players will allow them to enjoy the game regardless of how they like playing it, whether it is against the computer or against other players. The single-player requirement will need the computer AI to have different skill levels to accommodate different skill levels of the players. The multiplayer aspect of the game will need for there to be a casual matchmaking system, where players of all skill levels can play against each other, as well as a competitive matchmaking system, where players will be matched against players of the same skill level. The competitive matchmaking system must be set up in a way that changes a player’s skill ranking (ELO) upwards or downwards depending on if they win or lose games respectively. Additionally, there will be a tournament mode where brackets are set up and players move forward into the bracket when they win matches. It will also be necessary to have a system that has a record of a player’s win/loss statistics for the different games modes. A player must be able to choose to make their win/loss statistics public or private.

For the system’s data requirements, there will be a need for different databases, servers, as well as usages of services. There will need to be a database that will hold player information such as their username, friends list, win/loss records and their credit card information only if they choose to save it. Another database will need to exist to hold previous match information which will include all the players involved in that match, how long the game took to complete, and which player was the winner. A set of game servers will need to be running in order to host online multiplayer games and tournaments. An important service for the game will be the Google Maps API, which be utilized when new games are setup.

In order for the game to be set up properly, the classes that are needed as well as a brief description are as follows:

* Match: Object which handles interactions during gameplay such as passing turns to the next player and handling their moves with the other game objects.
* Player: Object of an entity interacting with the game. Subclassed by Human and AI classes.
* GameMap: Singleton object resembling a graph data structure. Consists of intersections as nodes. Only one instance per match.
* Street: An object resembling a physical street on a map.
* Intersection: Object resembling a node a node in a graph. Contains 2 or more Street objects.
* TerritoryCard: An object given to a player after they completed a successful attack phase.

The game will also utilize the following interfaces with the following functions

* PlayerFunctionality:
  + attackIntersection()
  + reinforceIntersection()
  + endTurn()
* GameFunctionality
  + rollDice()
  + checkPlayerEliminated()
  + checkForWinner()