Content -

Assassins

Requirements -

- Projected users - millions, in hundreds of thousands of games.
- Matchmaking - Players must be able to find 'open' games as well as games to which only invited players can join.
- Mobile interactivity required (phones used to verify kills via geolocation).
- Phones should be able to detect 'safe' zones and disallow play there
- Messaging - Players should be able to message each other, both anonymously and nominatively.
- Up to 20 players per game.
- Support a variety of rules variants

Later phase requirements:

- Video augmented reality 'kill shots', replacing Nerf guns or water balloons;
- Leaderboards
- Daily messages with news and information updates

Game Design -

The gameplay is based on a popular party game, Assassins, where people are randomly selected to be Impostors or Assassins who eliminate other players. Other players are known as Crewmates. Crewmates will have to identify the Assassins or complete all the tasks assigned to them to win the game. While Assassins have to kill or eliminate all the crewmates to win the game.

Assassins can use weapons to affect the game. They have three types of weapons in the game which cause different types of effects on the state of the game.

An Assassin will have these types of weapons available to them so the elimination of crewmates can be of these types -

- Direct elimination - Assassin uses a mock weapon, like nerf gun or water gun to eliminate a crewmate permanently from the game.
- Indirect elimination - Assassin sets up a trap and uses a weapon like poison, or a bomb. These weapons will be activated when a crewmate comes near them and activates it.

There is sometimes a third type of weapon in some variations of the game, using which an Assassin can capture a crewmate. We are not considering this variation for the application currently.

Map - The map will be based on the real world as the playing environment in a way similar to Pokemon Go. There will be safe zones where game is not allowed there. The host while creating a lobby selects how big the radius of the map will be.

Rules - The host player will be able to select the rule variant of the game that they want to play. General rules of the game will stay the same. But there will be an option to make the map small for the host, select the length of gameplay, and choose which game mode they want to play.

Game Modes -

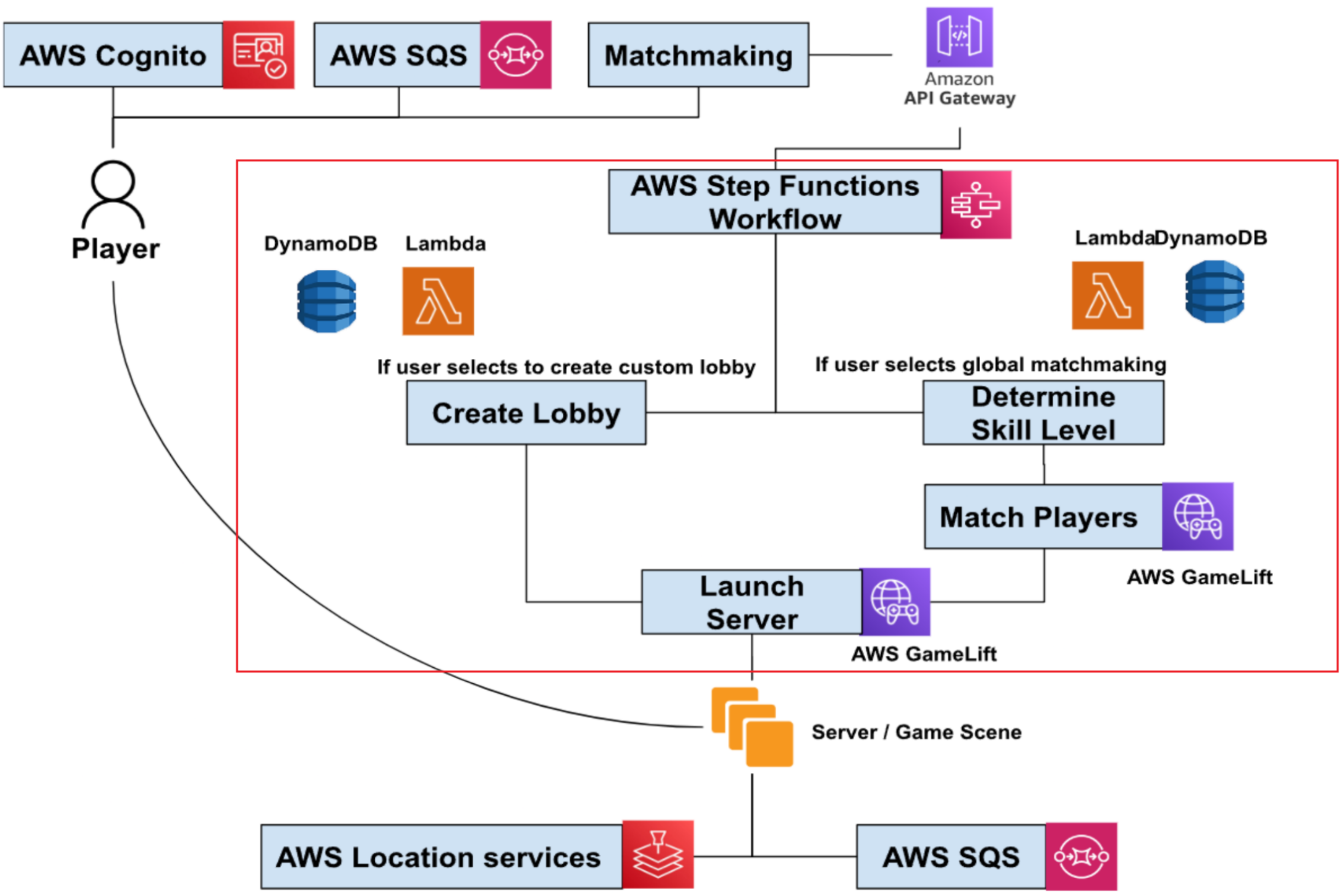
There will be three game modes offered -

- Classic - This mode will have crewmates trying to complete their jobs and identify Assassins, while the Assassins try to eliminate them. Last one remaining wins.
- Hide and Seek or Survival Mode - In this mode the identity of Assassins will be announced at the start of the game. Crewmates try to avoid getting assassinated.
- Zombie Mode - In this mode, assassinated crewmates turn into Zombies. Zombies try to convert remaining crew members to Zombies while they try to avoid them.

Use Cases -

- Players can send messages to other players on the home screen.
- Players can create a custom lobby.
- Players can invite his friends to their lobby.
- Players can use the global matchmaking option to play with other players of similar skill level.
- Players can send or broadcast messages to other players in the game anonymously and nominatively.
- Players can eliminate other players and that can be confirmed using geolocation.
- Players can not continue gameplay inside 'safe' zones.
- Players can vote and eliminate who they think is the Assassin.

Backend Architecture -



The above diagram shows the backend architecture of our Assassins gaming application. AWS Cognito is used to authenticate players, AWS SQS(Simple Queue Service) can be used to send and receive messages by player to other players. When a Player selects matchmaking, Amazon API Gateway invokes a parallelized AWS Step Functions workflow.

MatchMaking -

If the player selects they want to create a lobby, then AWS Lambda is used to create and manage the lobby, the details of this lobby are stored in the AWS DynamoDB database.

If the player selects global matchmaking, then their skill level is determined so that they can be matched with other players with similar skill level. The AWS Lambda function fetches player data from the Amazon DynamoDB database. Now, AWS Gamelift service queues and matches players based on their skill level.

Then AWS Gamelift is again used to launch the actual game server. This is the last step of the parallelised AWS Step functions Workflow. Now the details of the Server are passed back to the player through the Amazon API Gateway.

This way the player can connect to the server, which is indicated by a curved arrow in the diagram. If the player had created a custom lobby then the invites can be sent to other players using AWS SQS service.

Server/Game Scene -

The launched server is where all the gameplay takes place. Now when the players are in an active session, services such as AWS Location Services can be used to identify 'safe' zones where game is not allowed. It can also be used to verify kills using geolocation.

By using AWS SQS service we can enable messaging between players. They will have the option to send messages to other players anonymously and nominatively.

AWS Services used -

To implement our solution we used the following AWS services -

Amazon DynamoDB:
I used DynamoDB service to store information about players, and also stores a different table about game lobbies. If it is an open or a private lobby.

AWS Lambda:
AWS Lambda is used to do many tasks in the architecture. Fetch player data, calculate their skill level, to create a custom lobby and then add the details of this new game to the open game lobby table, when a player does global matchmaking then lambda function provides a list of open games from DynamoDB.

AWS SQS:
AWS SQS is used for messaging service among players, globally and in a lobby, and both anonymously and nominatively.

AWS GameLift:
I used AWS GameLift service for the task of queueing and matching players. This service is provided by AWS for making it easier for game developers to set up their gaming backend.
AWS GameLift is also used to launch game servers which are the computation machines in the architecture..

Software ilities -

- Scalability - For this app we know it needs to handle millions of daily users, so scalability is very important for our designed backend architecture.
- Responsiveness - is very important for any gaming app. Players need quick latency free responses to their moves in the game so they can react and make decisions. This is important to ensure a good user experience.
- Redundancy and Reliability - is important to maintain copies of data related to each player using AWS DynamoDB. There should be a reliable mechanism to keep track of their inventory, which can be purchased from in-game stores(in-app purchase) or earned in-game(pickup or loot).
- Adaptability - The game design needs to be altered and changes should be made to the game to offer exciting new content to the players periodically to keep user engagement high.

Possible Failures -

- A Player could have low ping which could result in poor performance and result in a bad experience even for other players. So to avoid this we will only allow players with similar latency to be able to match with each other and a maximum ping value which is allowed to be able to play the game at all.
- To avoid loss of player progress and inventory details we must keep ample backup of the tables stored in the DynamoDB database.

References -

https://en.wikipedia.org/wiki/Among_Us
https://en.wikipedia.org/wiki/Assassin_(game)
https://www.youtube.com/watch?v=vitoxDcE70o
https://aws.amazon.com/blogs/big-data/powering-gaming-applications-with-amazon-dynamodb/
https://www.youtube.com/watch?v=q-0yzA3WgXc
https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html