KnightVille

Avalon-like online multi-person game

Team Padpetpaloped

Members

Nithi Piyaphonnarinthon	62070503430
Romtam Tanpituckpong	62070503444
Wagee Jr. Nanta Aree	62070503445
Chatchapon Sukitporn-udom	62070503455
Kantawit Chatdamrongmongkol	62070503457

Abstract

This document contains all of the components of the final project of CPE327 Software Engineering, such as processes, ideas, and details. Our project is an Online multi-person game, based on “Avalon” which is a famous board game. In our project, we change from a board game to a 2.5D game perspective. All the rules and how-to-play we refer to the original Avalon board game.

In KnightVille, you can host the lobby, by running the server, and join the room using the host’s IP address. A match can contain 5 up to 10 people. An optional role can be selected and the number of optional roles can be picked depending on how many people on match setting are selected. In the lobby, players can change their name and skin they use which a player can see the change in real-time, and every player can see when a player saves his/her data. Once the players in a lobby are full, according to the number of players in a Setting game, then the host can start a match. During the game, besides the original gameplay for Avalon, players can walk freely on a map. For the menu of the game, there is an Option button that allows you to set the volume of BGM and Sound effects in our game. And can chat to others both during the game and in the lobby.

Appendix

Problem Definition

What is Avalon?

Avalon is a social deduction board game whose concept is based on A King Arthur theme. The games are played with 5 - 10 players, for which a certain number of players need to hinder the assignment to fail, and the rest need to work together and reveal the undercover agent working against them. The player will be separated into 2 teams, a Good team and an Evil team. When selected to do the assignment, Good team's members can select the assignment to be passed only, while the Evil team's members can select both passed and failed.

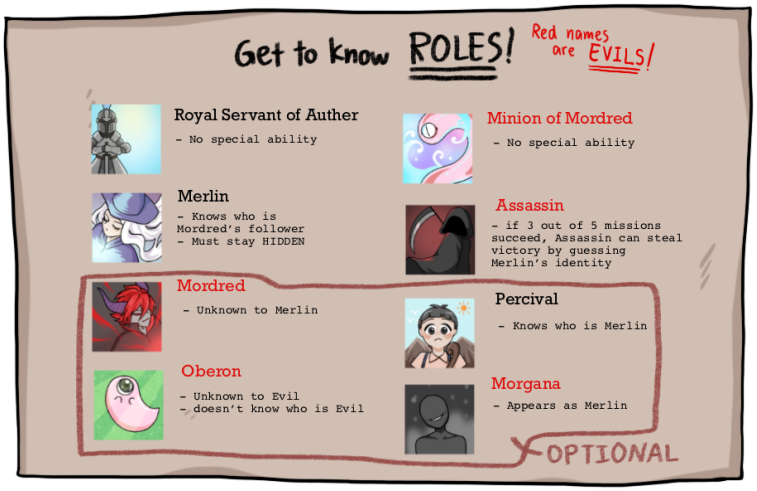


Fig.1

There are 8 roles in Avalon, half are standard roles and the other half are optional roles.
The standard roles are Royal Servant of Arthur, Merlin, Minion of Mordred, Assassin.

- **Royal Servant of Arthur** is a person in a Good team. This role has no special ability.
- **Merlin** is a person in a Good team. Merlin has the ability to know who is in the Evil team, but not know the role. The only role Merlin can see is Oberon, however, Merlin will see Mordred as a teammate. The most important part of this role is the player needs to stay "hidden" until the end of the game.
- **Minion of Mordred** is a person in an Evil team. This role is similar to Royal Servant of Arthur but it is on an Evil team.
- **Assassin** is a person on an Evil team. At the end of the game, if more than half the assignment is passed, Assassin has a chance to kill Merlin. If an Assassin selects a Merlin correctly, an Evil team will win. Otherwise, a Good team wins.

The optional roles are Oberon, Mordred, Morgana, and Percival.

- **Oberon** is a person on an Evil team. Although Oberon is in an Evil team, Oberon does not know who his or her teammates are, and the teammates also do not know if he is Oberon. Furthermore, His/Her role can be seen by Merlin.
- **Mordred** is a person in an Evil team. Mordred is comparable to the head of an Evil team. His role cannot be seen by Merlin.
- **Morgana** is a person on an Evil team. Morgana can disguise as a Merlin which will make Percival confused about who is the real Merlin.
- **Percival** is a person in a Good team. Percival is Merlin's disciple so He can see who Merlin is. However, Percival also sees Morgana as a Merlin.

Avalon rule

Fig.2

It will separate into 2 teams - good and evil. Either a good or evil team will win. (win as a team, not individual)

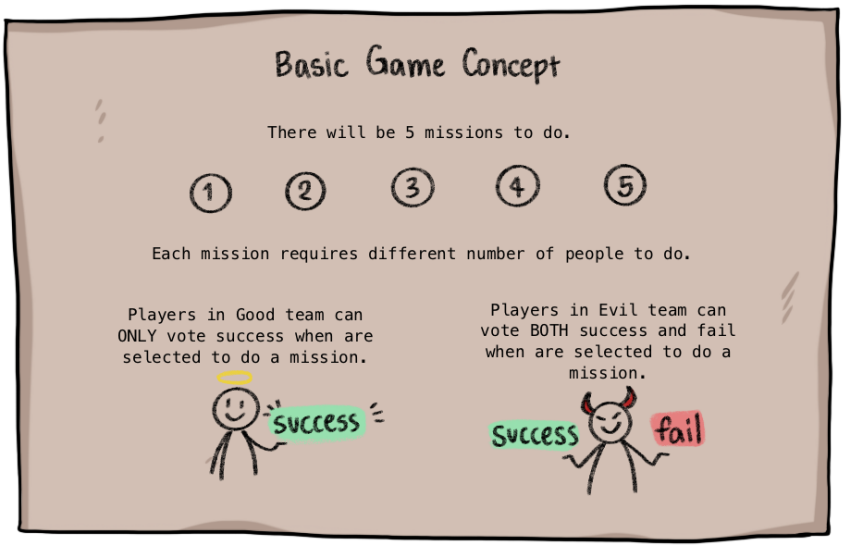


Fig.3

Fig.4

There will be 5 missions to do. In order to win for the good team, three missions need to be done successfully and Merlin is not killed at the end of the game. In order to win for the evil team, three assignments need to be failed.

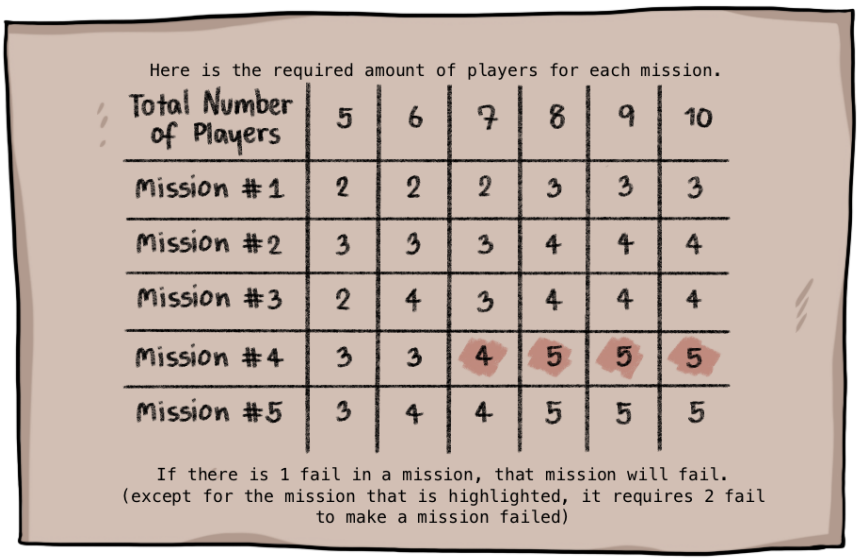


Fig.5

Steps to play

Fig. 6

1. Each player will receive a role by random, and a play will be selected to be a team leader.
2. The team leader selects a certain amount of players to do the mission. Leader can select himself/herself to do the mission.

Fig.7

3. After the team leader selects who is going to do the mission, everybody can vote to accept or reject this team.
4. If the "accept" > "reject" then this team can do the mission.

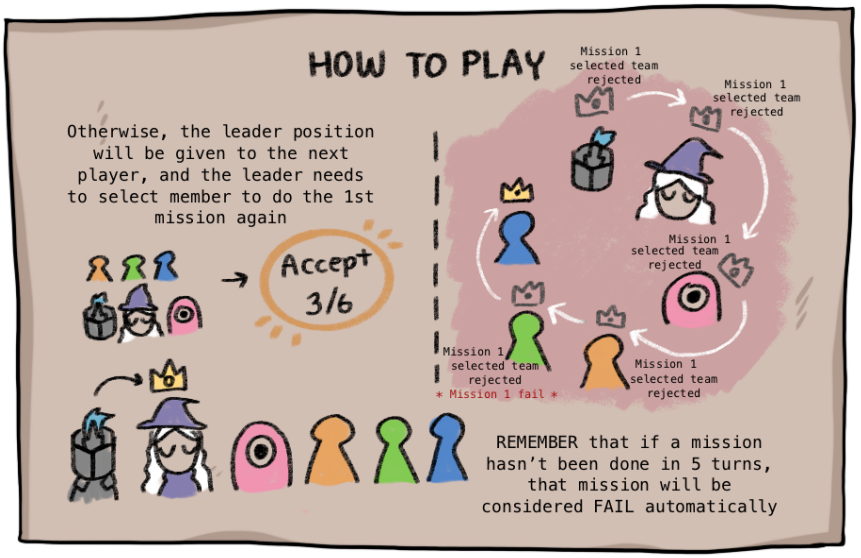


Fig.8

Otherwise, the leader position will be given to the next player, and the leader needs to select members to do that mission again.

The most important thing is that if a mission has not been done in 5 turns, that mission will be considered to fail automatically.

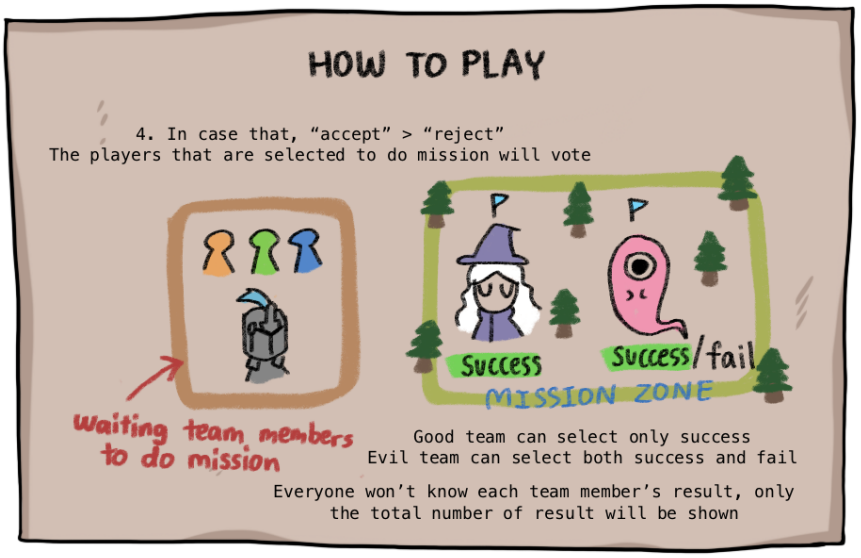


Fig.9

4. In case that, “accept” > “reject”, the players that are selected to do the mission will vote. A good team can select only success and an Evil team can select both success and fail. Everyone will not know each team member’s result, only the total number of results will be shown.

Fig.10

5. Once the result of the mission comes out, the leader position will be given to the next player and all steps are repeated over and over again until the game ends.

Requirements

Lobby phase - The lobby waiting for other players to join in.
Start phase - The actual game after the host starts the match.

Functionality

Must have:

- Game must allow players to quit the game via the menu.
- Game must allow players to create and customize their match (this player will be the host).
- The match can be customized to contain 5 - 10 players and be able to add a special role (optional role). The number of special roles will depend on the number of maximum players that have been selected.
- Game must allow players to join other player's matches via the server's IPv4 address and Port.
- Game must allow players to initialize their name and appearance.
- Game must allow players to walk around and communicate with each other via chat box during the Lobby phase and Start phase.
- Game must allow the host player to start the match anytime when the number of players (including host) reach the maximum number of players in the setting.
- Game must allow players to make a choice corresponding to their role in each game state during the Start phase.
- Game must have the ability to show the result and determine the result according to the phase of a game during the Start phase.
- If one of the players left the game during the Start phase, the game must have the ability to handle this situation by bringing every player back to the Lobby phase.
- Game must provide a warning message popup to handle the invalid input given by players.
- If the host of a match leaves or disconnects from the game, the game's server must have the ability to change host position to the next player within the match.
- Game's server must be able to automatically randomize and assign roles to each player when the match starts.
- Game's server must allow any players to run and allow any player to host (No need to be only the player who runs the server that can host, anyone can host if they can connect to the server).
- Game's server must allow a player to change IPv4 and Port that are used to connect the server via Server config file (in case player wants to use virtual local network IPv4 address not their own local IPv4 address).

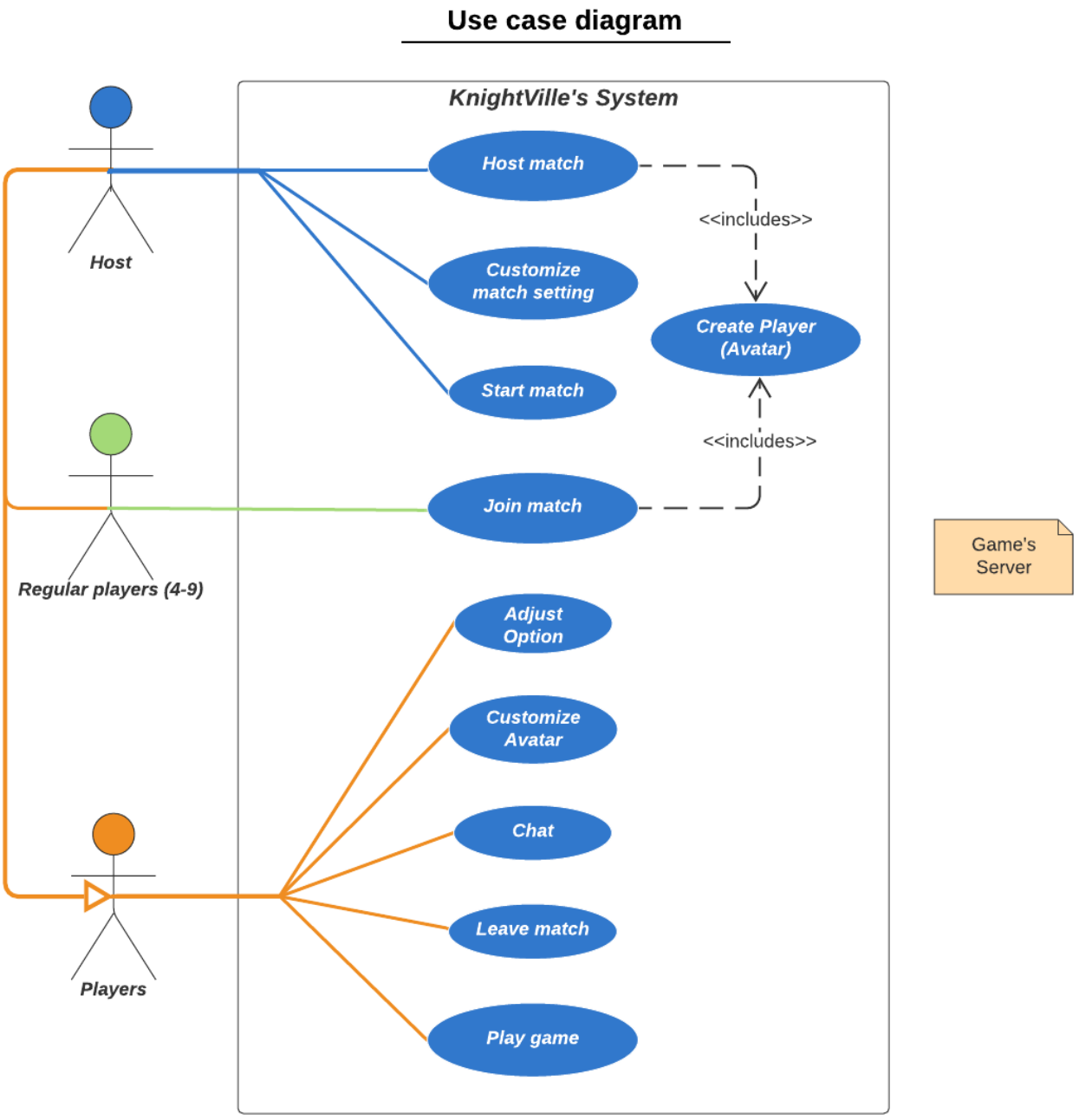
Good to have:

- Game must allow players to customize the background's music and sound effects in their game.
- In the lobby, the game must allow players to leave and customize their names and appearance. Host can also customize match settings during this phase but only a special role (optional role) that can be customizable, not including the maximum number of players.
- Game's server config file must be able to recover if it's damaged. The server must be able to rewrite it with default setup (local IPv4 address with port 5555).

Constraint

- Game required Python 3.7.9++ and Pygame version 2.0.1 or more.
- Game must run smoothly on the Windows 10 operating system.
  Not guaranteed on Linux/UNIX, macOS or other OS but for other versions of Windows OS, it depends on the version of Python and Pygame.
  You can check by link below:
  Python: https://www.python.org/downloads/windows/
  Or search by "Python Releases for Windows"
  Pygame: https://pypi.org/project/pygame/
  Or search by "pygame - PyPI"
- Game requires a CPU that can run multiple threads at the same time.
- Game's server must be run separately from the actual game.
- Game's server must work fine with a local network or virtual local network e.g. Hamachi, Radmin, etc.
- One server can host only one match at a time but allow to have multiple servers on multiple computers (one server per computer).
- Game must use only an IPv4 address with TCP (Transmission Control Protocol) in order to make a connection.
- Game must run with a screen size of 1280x720 only.
- Game's files must be perfectly undamage in order to run the Game properly.

Use case diagram



KnightVille has only one main actor which are Players. Players can then divide into 2 actors which are Host and Regular players that can be only 4 to 9 players at a time (not including host). Each actor has different responsibility in each use case as shown in the diagram.

Since many use cases are involved with Game's server, we decided not to draw all the lines into the diagram.

Brief explanation of each use case

- ★ **Use case that involves with Game's server**

[Not require precondition]
Host match : Connect to server by address then initialize match setting to start a match.
Join match : Connect to server by address and join current existing match.

[Precondition : Players already in the lobby]
Start match : Host can start the match whenever player reaches maximum player.
Customize match setting : Host can adjust the setting of current match with some limited options.
Customize Avatar : Every player can customize their avatar name and appearance apart from their initialization.
Chat : Every player can communicate with each other by using a chat box.
Leave match : Every player can leave the match anytime while the match doesn't start.

[Precondition : Match already started by host]
Play game : Actual gameplay involves many states, rules, and choices.

- ★ **Use case that doesn't involves with Game's server**

[Not require precondition]
Adjust Option : Every player can customize their own game from the given option to their liking.

[Precondition : Players already join the match or host match]
Create Player : Initialize name and appearance for user avatar in current match.

Use case narratives

Use case	: **Host match**
Actor	: Host
Goal	: Create a lobby
Pre Condition	: Not require
Post Condition	: Lobby joinable

Main Success Scenario :

1. User launches the server file to be the game host.
2. Host is on the menu.
3. Host click host on menu.
4. Popup to ask his/her IP address and port
5. Host enters his/her server ip address and port.
6. Host click connect.
7. Lead host to match setting.
8. Ask the host to set a number and role.
9. Host decides the number of players in the match.
10. Host selects a role that he/she wants to appear in the match.
11. Host click create lobby.
12. Lobby successfully created.

Extension scenario (a)

4a. Host clicked connect
5a. Server is not launched yet.
6a. Display error popup contains the error message.
7a. Host close popup.
8a. Return to step 2.

Extension scenario (b)

4b. Host click connect
5b. IP is wrong.
6b. Display error popup contains the error message.
7b. Host close popup.
8b. Return to step 2.

Extension scenario (c)

9c. Host click create lobby button.
10c. The lobby with the same server IP address already exists.
11c. Display error popup contains the error message.
12c. Host close popup.
13c. Lead Host to main menu.
14c. Return to step 2.

Use case : **Join match**
Actor : Regular Player
Goal : Join lobby
Pre Condition : Not require
Post Condition : Successfully to connect a lobby

Main Success Scenario :

1. Player is on the menu.
2. Player clicks join on the menu.
3. Popup asks host ip address and port.
4. Player enters the Host's IP address and port.
5. Player clicks connect.
6. Success to connect host server.
7. Lead player to Create Player (User Case).
8. Lead player to lobby.

Extension scenario (a)

5a. Player clicks connect.
6a. Failed to connect to the host server.
7a. Display error popup contains the error message.
8a. Player close popup.
9a. Return to step 1.

Use case : **Create player**
Actor : Player
Goal : Player have multiple choice of avatar and can customize their ingame name.
Pre Condition : Joined the server.
Post Condition : Their avatar and custom name is displayed in the game.

Main Success Scenario :

1. Ask to select the avatar from the set that is currently in the game file.
2. Ask players to enter the ingame name.
3. Player clicks join.
4. Player ingame name is valid.
5. Lead players to lobby.

Extension scenario (a)

3a. Player clicks join.
4a. Player ingame name is invalid.
5a. Display error popup contains the error message.
6a. Player close popup.
7a. Return to step 2.

Extension scenario (b)

3b. Player clicks join.
4b. Server is closed.
5b. Display error popup contains the error message.
6b. Player close popup.
7b. Lead player to the menu.

Use case : **Adjust Option**
Actor : Player
Goal : Set the game volume(Music and Sound effect)
Pre Condition : Not require
Post Condition : The game volume is set.

Main Success Scenario :

1. Player clicks the Option button.
2. System shows the option page.
3. Player drags the controller box to set the music and sound effect volume.
4. System sets the volume according to the position of the controller box.

Use case : **Customize Avatar**
Actor : Player
Goal : Change the appearance or/and in-game name
Pre Condition : Not require
Post Condition : The player's skin or/and in-game name is changed.

Main Success Scenario :

1. Player clicks the knight puppet which is standing in the lobby.
2. System shows the pop-up for customizing the avatar.
3. Player changes name and skin.
4. Player clicks submit.
5. System changes player's appearance

Extension scenario (a)

2a. System shows the pop-up for customizing the avatar.
3a. Player does not change anything.
4a. Player click submit
5a. System lets players use the same in-game name and skin.

Use case : **Start match**
Actor : Host Player
Goal : Start matchmaking
Pre Condition : Players already in the lobby
Post Condition : All player is in match

Main Success Scenario :

1. The lobby is full (number of the players in lobby is equal to certain number in match setting)
2. Host clicks the start button.
3. Bring all players to the in game screen.

Extension scenario (a)

1a. The number of players in the lobby is less than the certain number in match setting.
2a. Host clicks the start button.
3a. System shows the error pop-up that the current number of players is insufficient to start the match.

Extension scenario (b)

1b. The lobby is full (number of the players in lobby is equal to certain number in match setting)

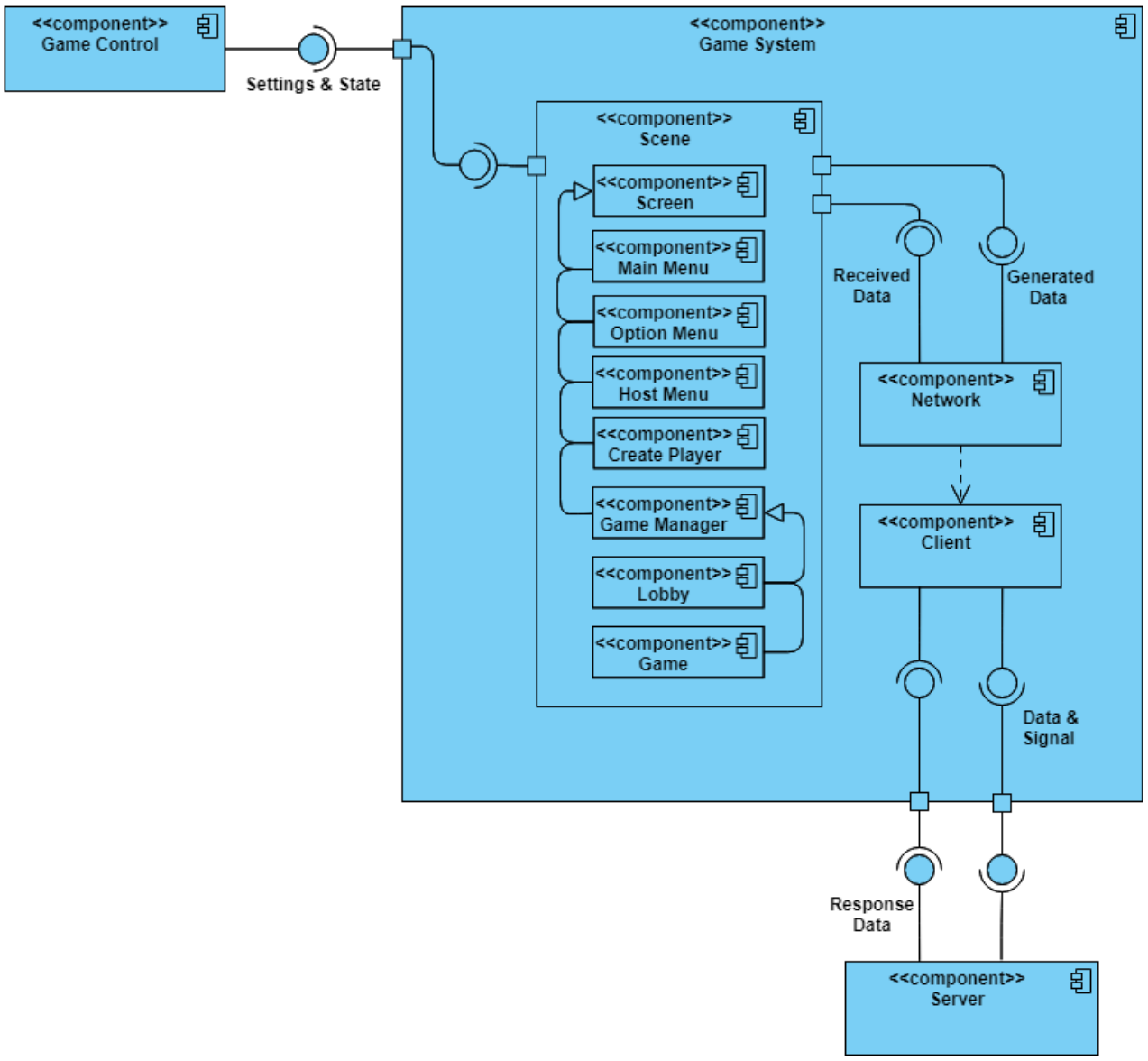
2b. Some players are customizing their avatar in the *Edit player* page but did not click the submit button yet.

3b. Host clicks the start button.

4b. System starts the match with the same initial in-game name and skin for the player who did not click submit.

Architectural Design

Component Diagram



KnightVille, in overview, can separate into three modules which are:

Game Control : Control game state and provide interface for all initial game settings.

Game System : Show overall components in game.

1. **Scene** : Group of components related to the game's scene. Every component in the Scene received settings and state from Game Control. And, you can see that each component in the Scene have arrow connected between them this arrow indicated inheritance between components in this case Main Menu, Option Menu, Host Menu, Create Player and Game Manager inherited most of the feature from Screen same as Lobby and Game that also inherited most of the feature from Game Manager. Most of the components in Scene required data from the network and also generated data for the Network as well.
2. **Network** : Provide suitable data and receive generated data for Scene. Dotted arrow indicates dependency which tells that Network have to depend on Client to receive and transfer data between Server.

3. **Client** : Sending and Receiving Data between servers via signals that indicate the server to perform a specific task. Unlike the Network, Client will perform only general tasks which means the format of send or receive data will not be prepared specifically for Scene so that's why Network is needed to perform this task.

Server : Local server contains some essential game data and all clients data. Server also manages all activity that occurs to the server for example Client will send signal along with data, the Server will know how to response to the data via signal then server will send Response Data to the Client for further use.

Networking

Our game uses a local network to make an online environment for our players. We use only IPv4 address and port number (default is 5555) both with TCP (Transmission Control Protocol) in order to make a connection.

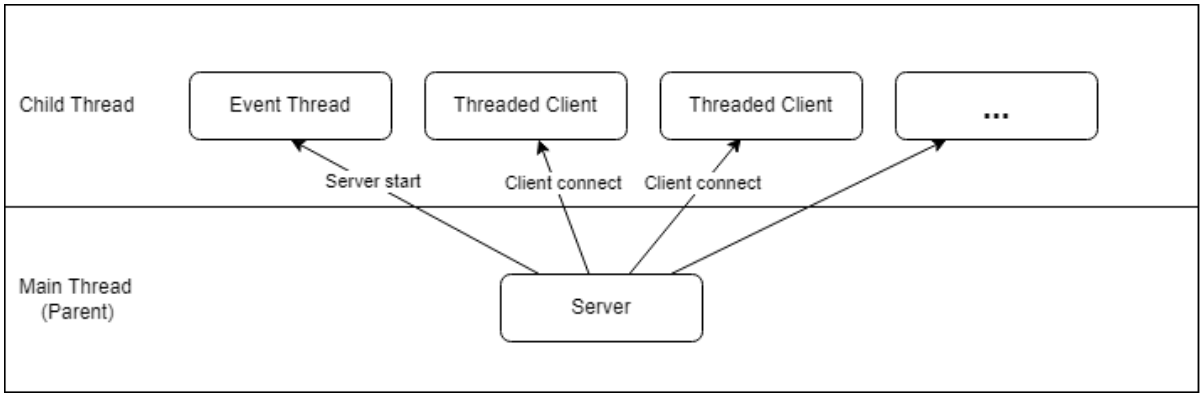
IPv4 (Internet Protocol version 4) is a 32-bit address for example 192.168.15.4 .
TCP (Transmission Control Protocol) is a transport protocol that is used on top of IP to ensure reliable transmission of packets. More info:
https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:transporting-packets/a/transmission-control-protocol--tcp

- **About Server**
  - The game needs at least one Server in order to play.
  - To reduce resource consumption and error during data transmission, Server must be run separately from the main game.
  - Server can be run on any computer within the same local network.
  - Server doesn't need to run on the same computer with the player who wants to host the game, players are allowed to run Server on any computer they want within the same local network.
  - To connect the Server, players need to join the Server with the same IPv4 and port as the Server.
  - Any players that currently connect to the Server can host the match.
  - One Server can host only one match.
  - Multiple Servers can be run at the same time on different computers or different IPv4 addresses (Recommended one Server per computer).
  - Players can change Server's IPv4 and Port number via "server_config" file.

Threading

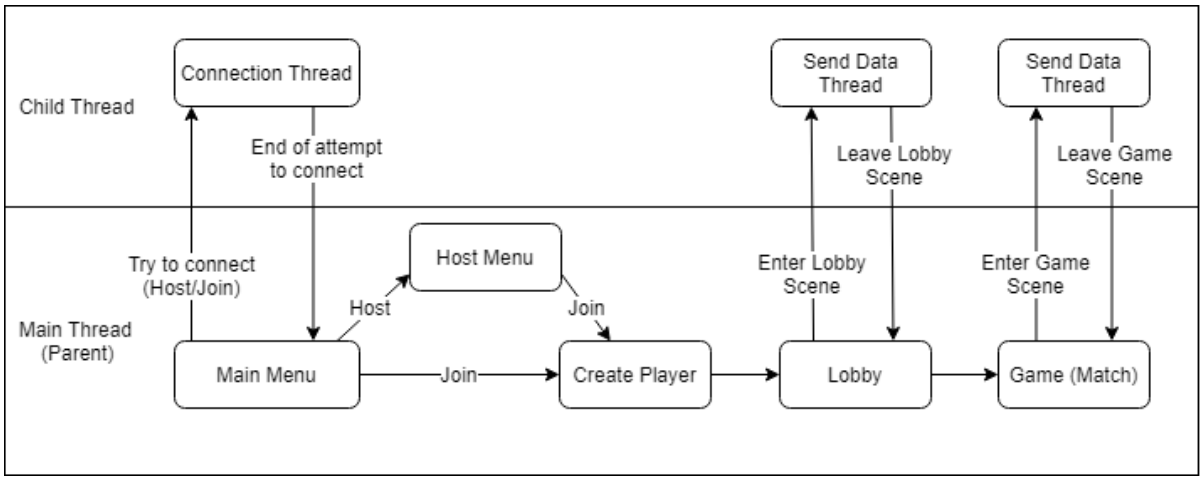
We use the concept of multithreading to acquire some advantage to our game in many ways as shown below.

Server



Server will create a thread called “Threaded Client” for every client that successfully connects to the server. Every thread will run parallelly in the same time (no synchronization), main thread (Server) will wait for any attempt to connect from other clients while each child thread (Threaded Client) will handle every request from their corresponding client. Maximum number of “Threaded Client” depends on the number of maximum players set in a match (usually 5 to 10). “Event Thread” will create immediately after Server is started, This thread will wait for the event to be handled when the match is started (handle in-game event).

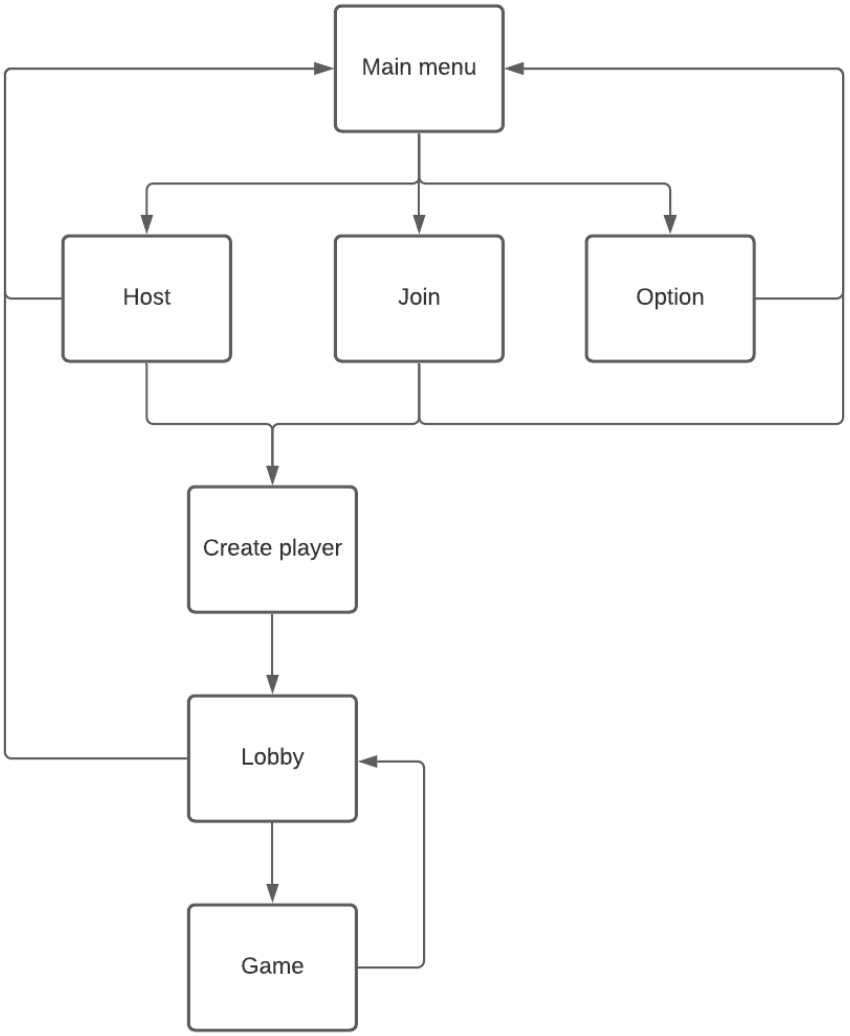
Individual Client (Each player’s game)



Note: This diagram shows only interactions between thread and state (scene) not the Navigation map.

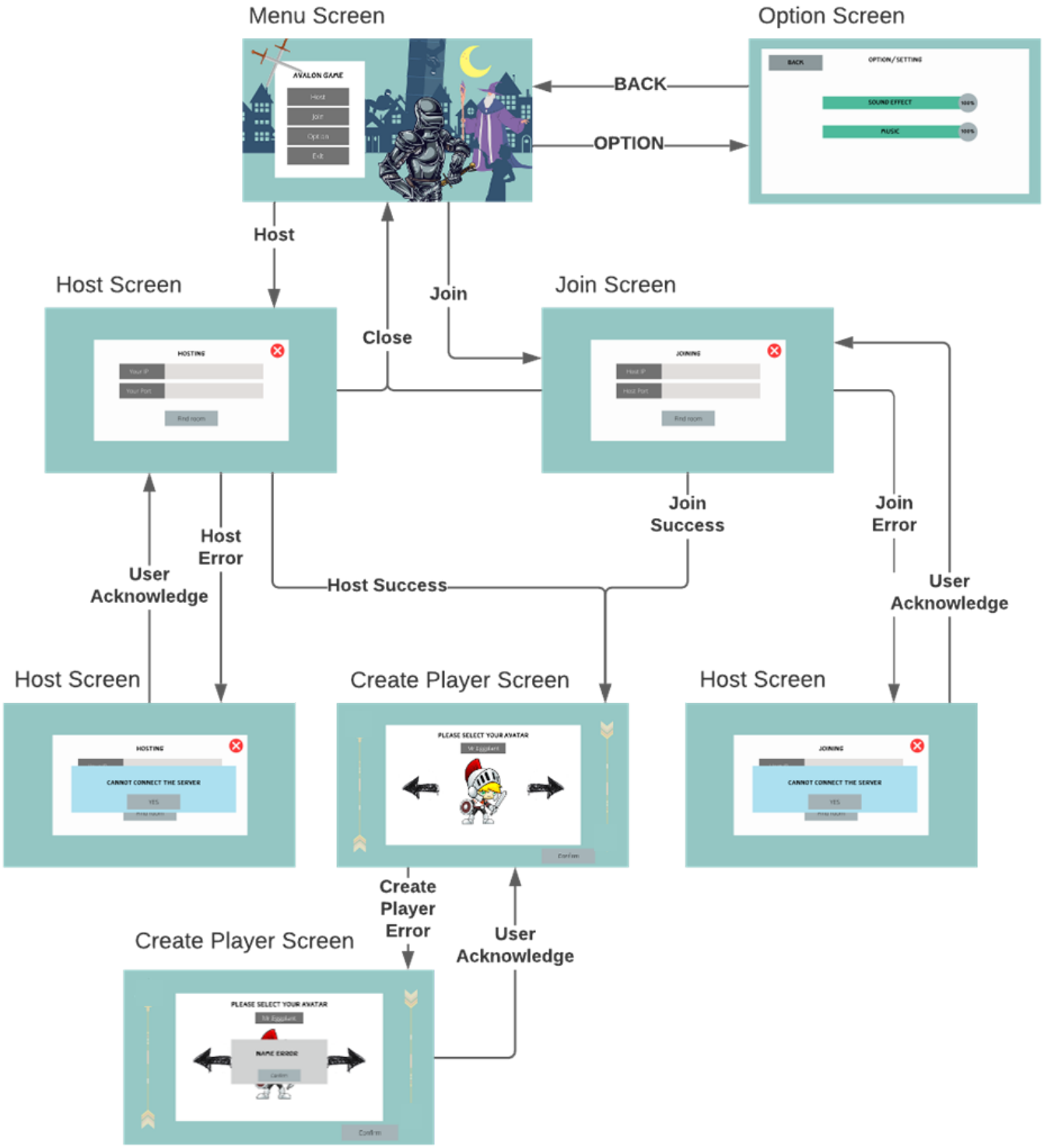
In each Client there are two main types of threads that help increase the game performance. First is the thread called “Connection Thread”, this thread will handle “the attempt to make a connection from client to server”. When a player Host or Join the match in Main Menu, “Connection Thread” is initiated to make a connection parallely with Main Menu. This prevents the Main Menu from getting struck while attempting to make a connection. “Connection Thread” will join back to the Main Thread automatically after it finishes its task. Second is “Send Data Thread”, this thread will run immediately after entering Lobby or Match. The purpose of this thread is to send data in the background while Lobby or Match are drawn on screen. “Send Data Thread” will join the Main Thread when the scene is changed. (More description in Sending and Receiving Data Method)

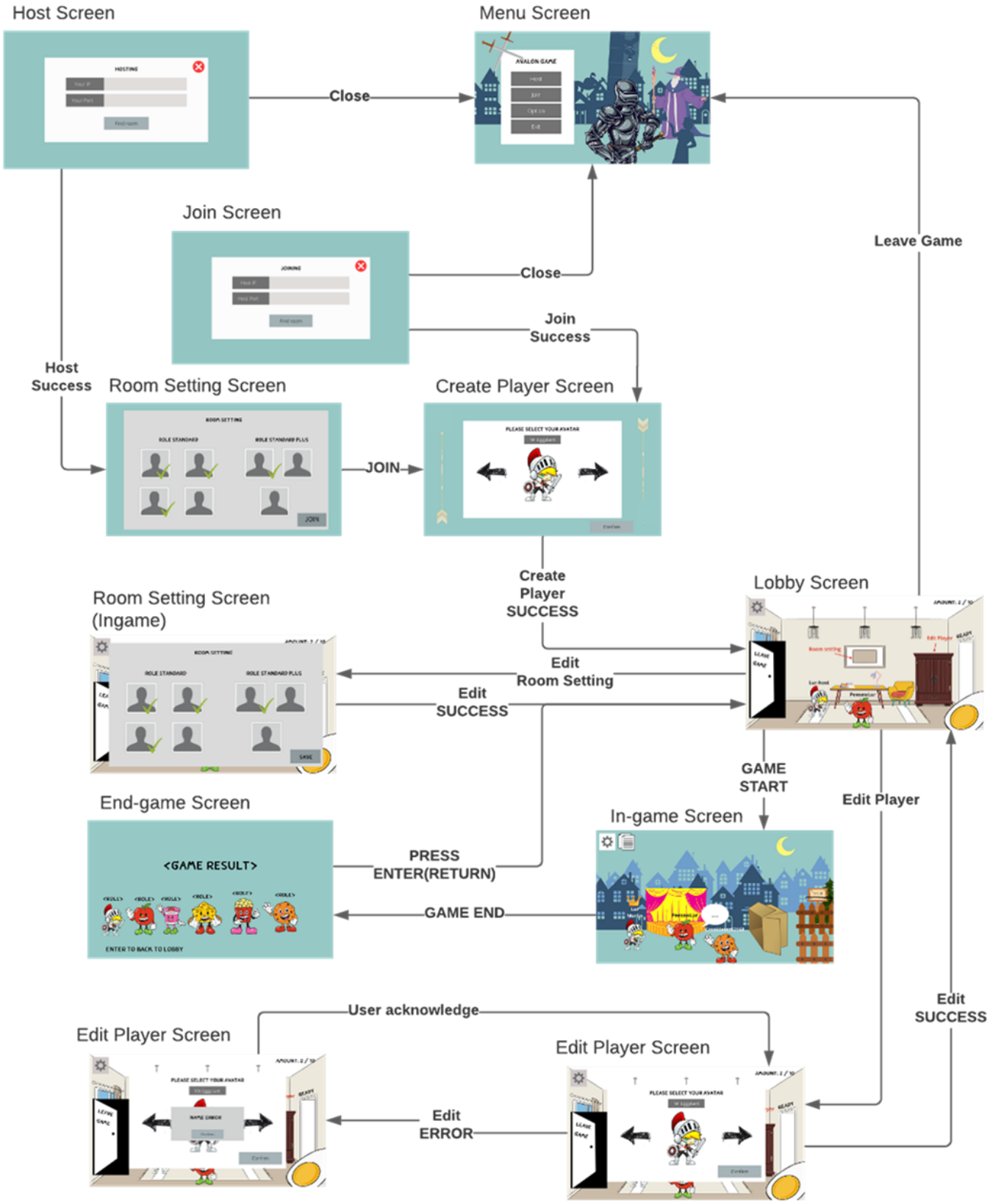
Navigation map



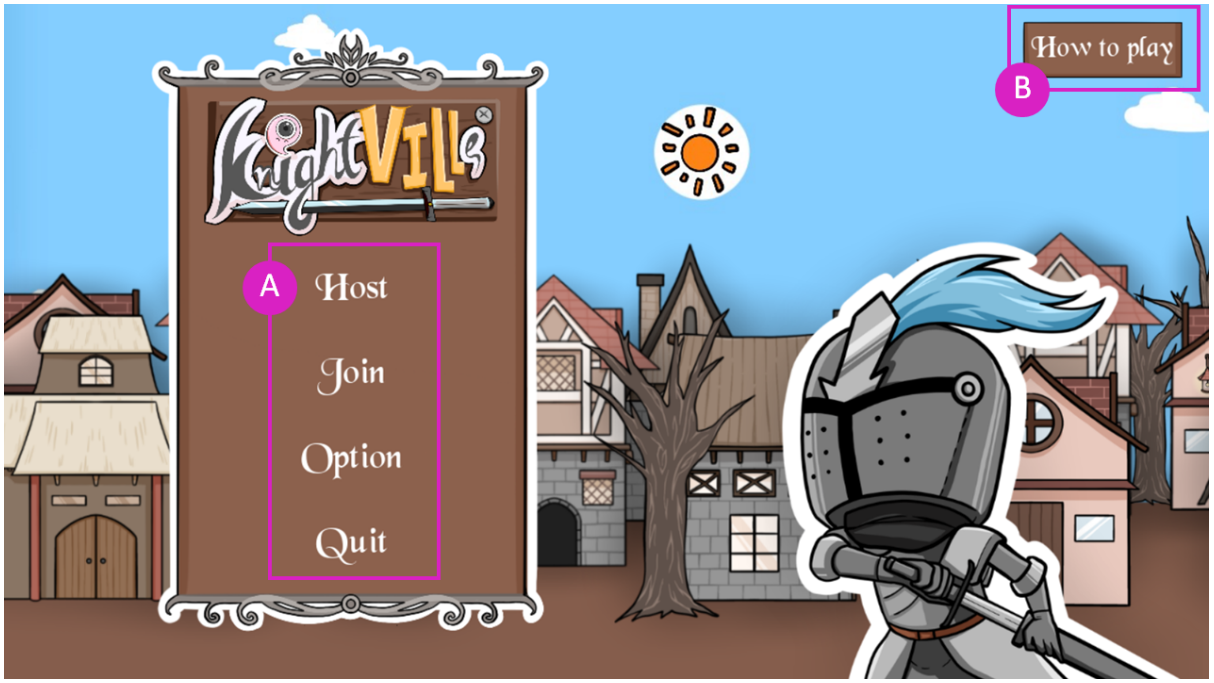
1. First, when you open the program it will start with the main menu, from that you can access three pages which are *Host*, *Join* and *Option* page. You can also quit the game from the main menu.
2. After you click the *Host* or *Join* button and enter the host IP address, the system will show the player creating page before going to the lobby.
3. For the in-game part, we would like to explain in the form of State diagram which you can find it in the In-game state explanation part.

UI mockups

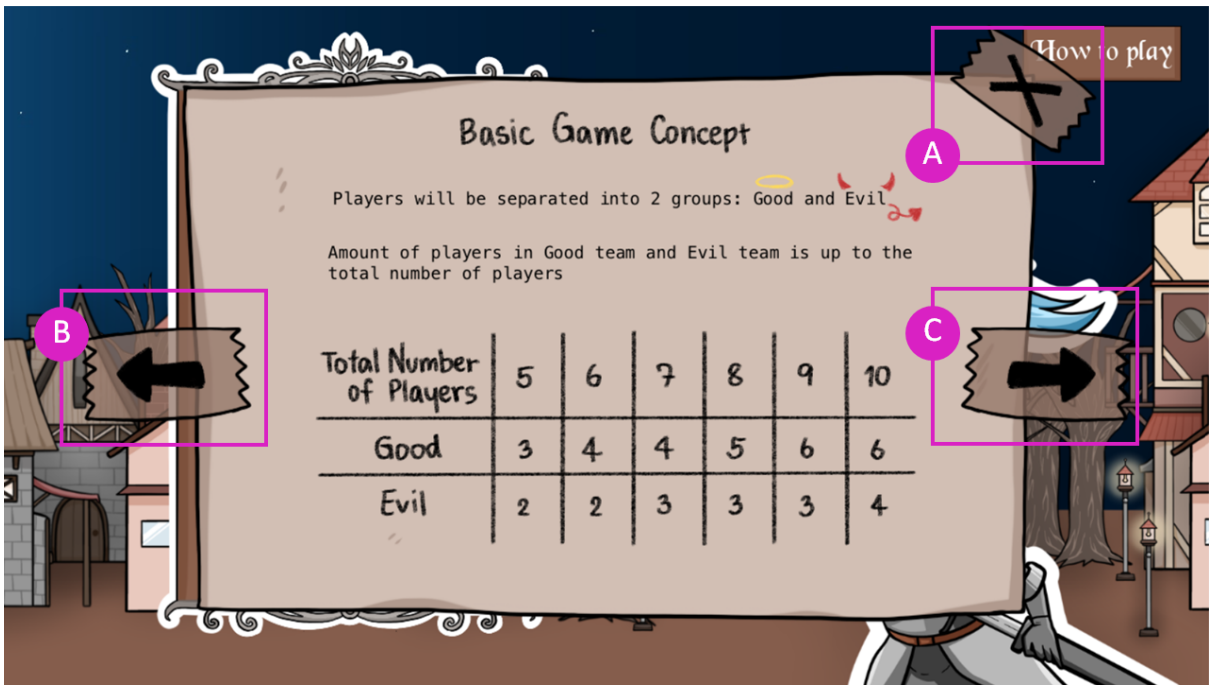

Menu Screen
Option Screen
BACK
OPTION
Host
Join
Close
Host Screen
Join Screen
Host
Error
User
Acknowledge
Host Success
Join
Success
Join
Error
User
Acknowledge
Host Screen
Create Player Screen
Host Screen
Create
Player
Error
User
Acknowledge
Create Player Screen


Host Screen
Menu Screen
Close
Join Screen
Close
Leave Game
Join
Success
Host
Success
Room Setting Screen
Create Player Screen
JOIN
Create
Player
SUCCESS
Lobby Screen
Room Setting Screen
(Ingame)
Edit
Room Setting
Edit
SUCCESS
GAME
START
Edit Player
End-game Screen
In-game Screen
<GAME RESULT>
PRESS
ENTER(RETURN)
GAME END
Edit
SUCCESS
User acknowledge
Edit Player Screen
Edit Player Screen
Edit
ERROR

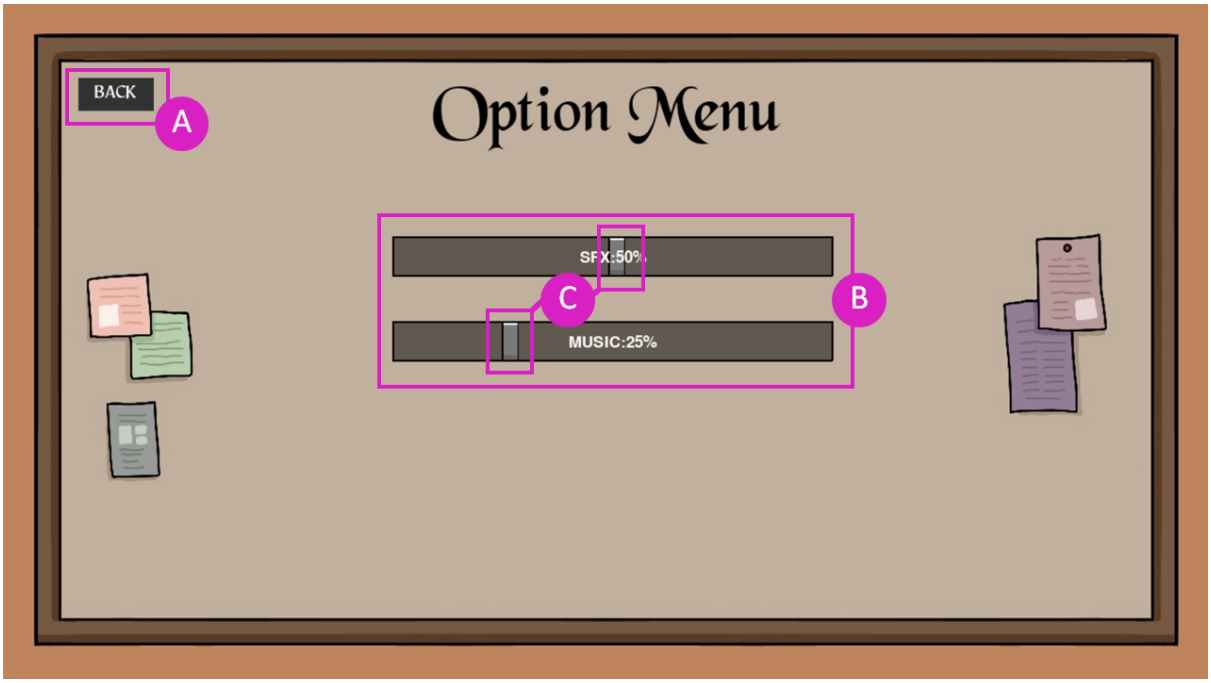
UI Callout



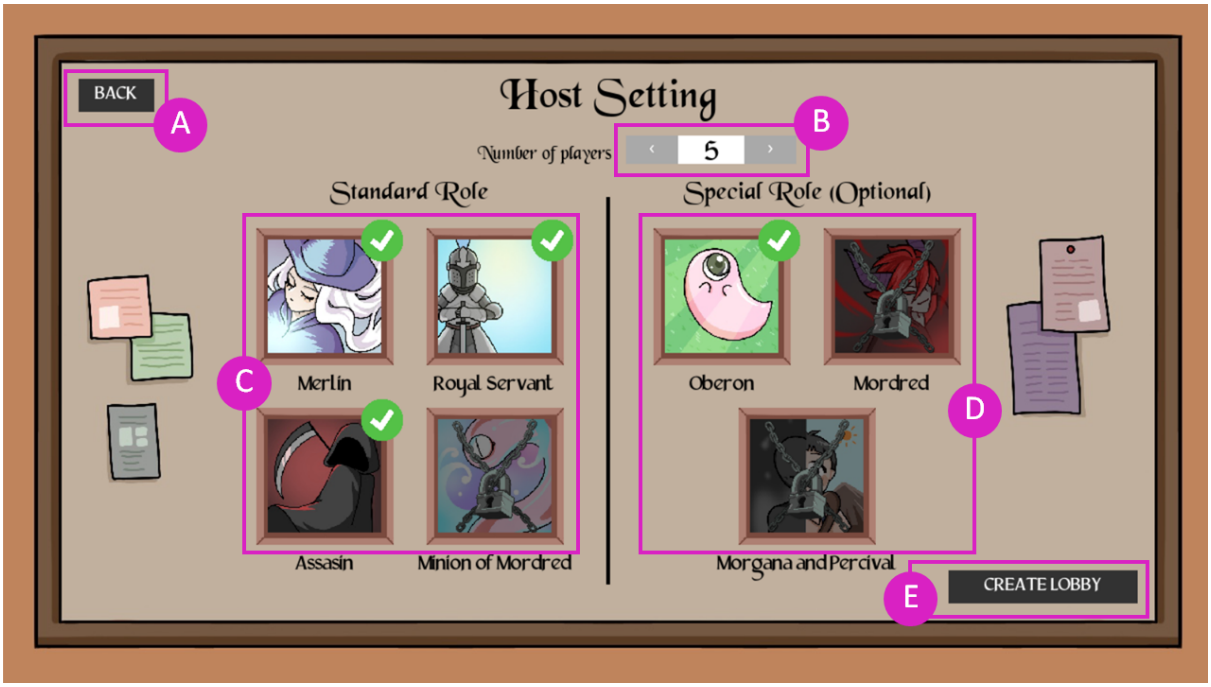
A) **Menu button :** Button that clicks to select that leads to the certain page or screen.
B) **How to play button :** Button to show how the game is play or the rule used in the game briefly.



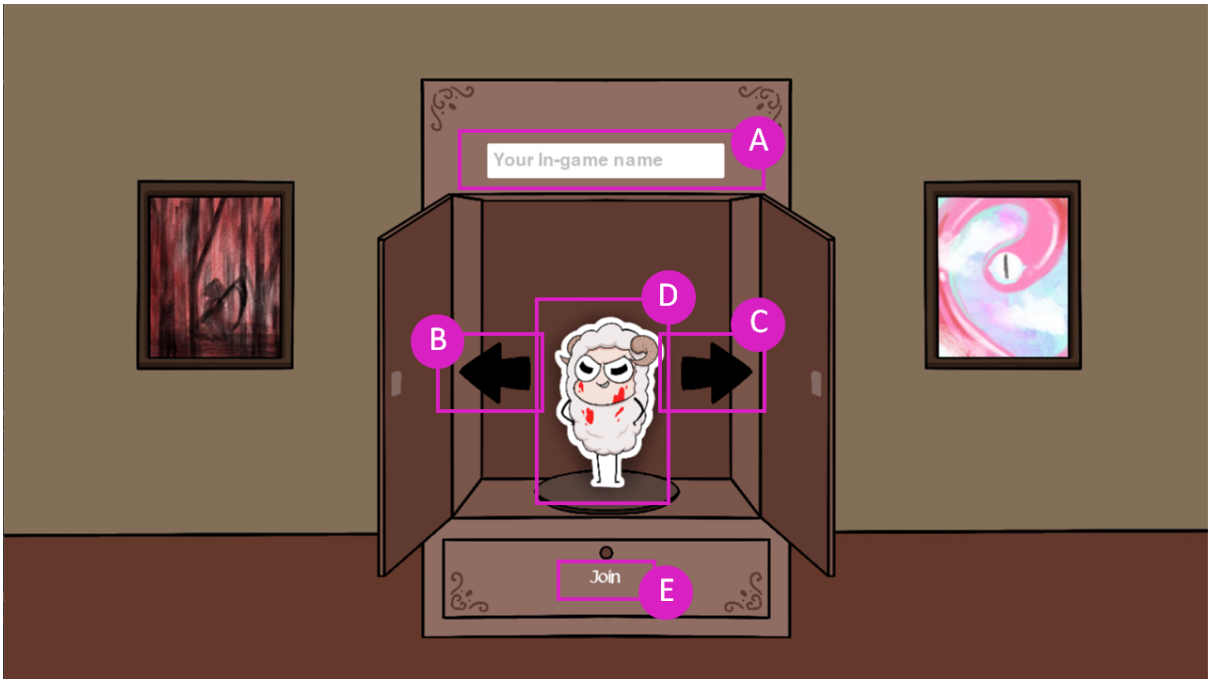
A) **Close button :** Button to close and go back to the menu.
B) **Previous page button :** Button to go to the previous page.
C) **Next page button :** Button to go to the next page.



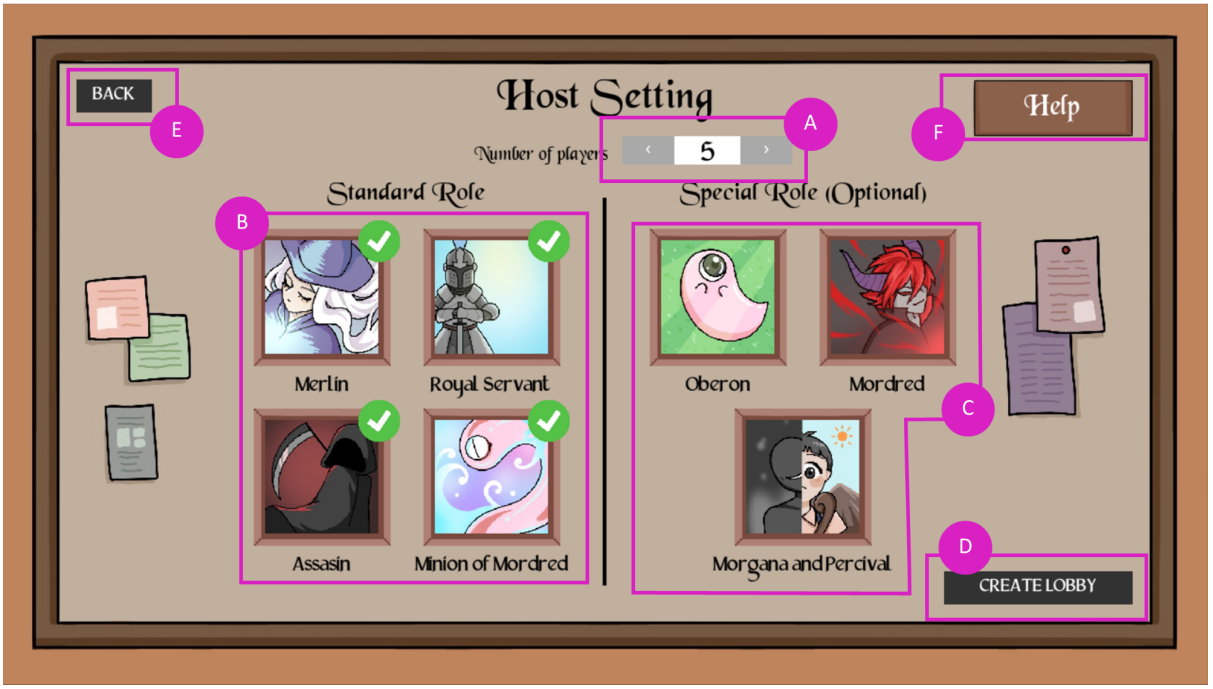
A) Back button : Button to go back to the menu.
B) Volume bar : Bar that displays the value of the volume.
C) Control volume : Click and draggable object that can adjust the volume.



A) Back button : Button to go back to the menu.
B) Player number setting : the area that asks the number of players of the match.
C) Standard Role : the area that displays the standard role that will appear in the game.
D) Special Role : the area that displays the special role that is optional to be selected or not and will appear in the game.
E) Create Lobby button : Button to save the role setting.



A) In-game name textbox : the area that asks to enter their in-game name.
B) Left arrow : Button to display the previous skin.
C) Right arrow : Button to display the next skin.
D) Skin display : the area that displays the skin that is currently selected.
E) Join button : Button to save the skin and in-game name of player and go to the lobby.



A) Number Player Setting : Where you can change the player number in the match.
B) Standard Role : Show the standard role that will be available in the match which can't be selected.

C) **Special Role :** Show the special role that will be available in the match which can be selected by click.

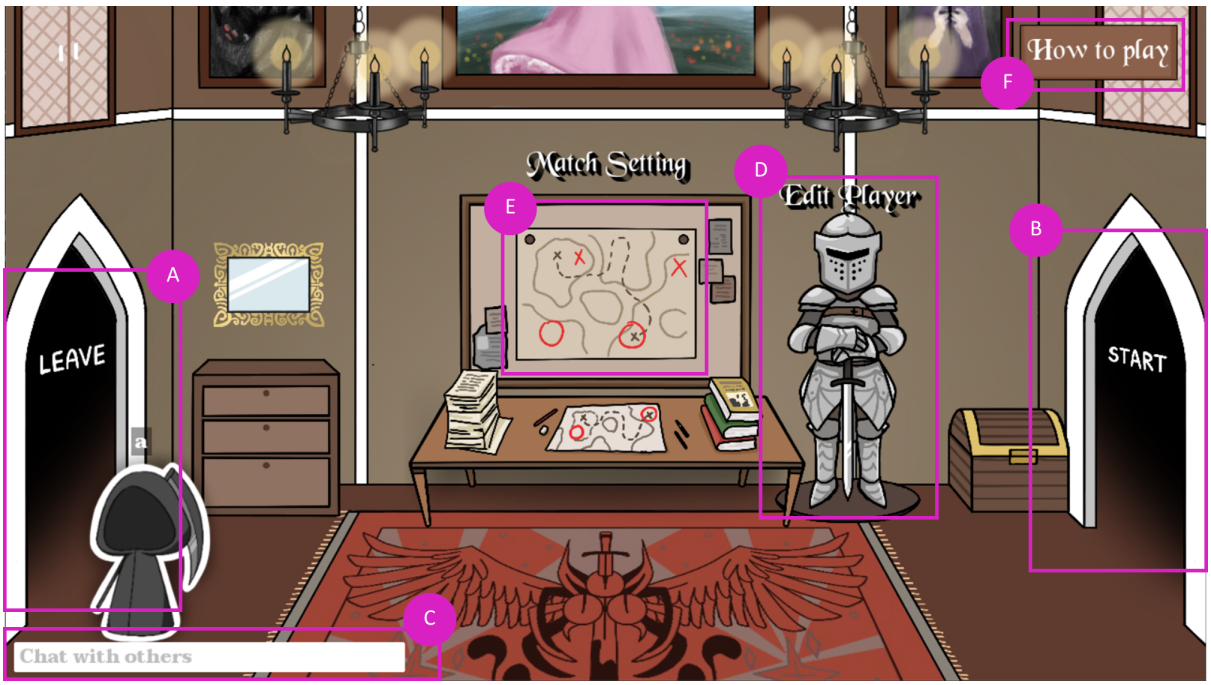
Not selected

Selected

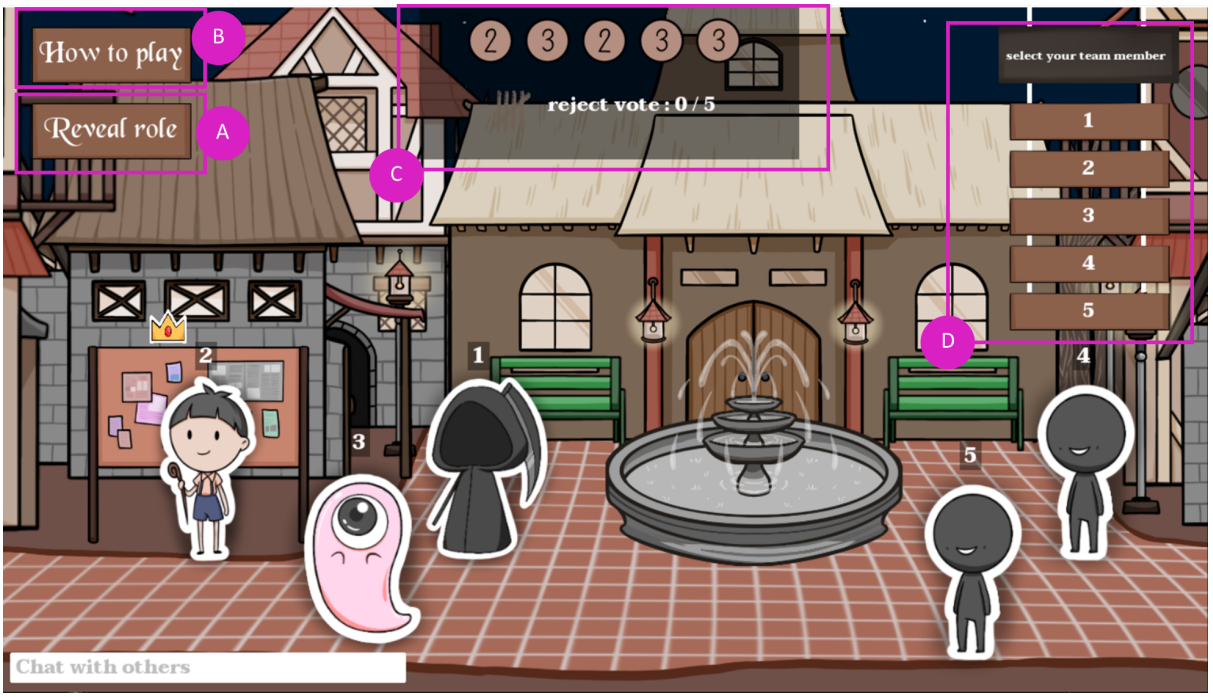
Can't Select

- If selected, a role will appear in the game. otherwise a role will not appear in the game if not selected.
- If there is a lock on top of a role, it means that roles being selected at the limit can be selected with the amount of players.

D) **Create lobby button :** Button to create a lobby which has the setting as User which hosts the match set.
E) **Back button :** Button to go back to the menu which will lose all current settings.
F) **Help button :** Show the detail and ability of all the roles.



A) **Leave button :** Button to leave the lobby and go back to the menu.
B) Start button : Button that only the host can click to start the match.
C) Chatbox : Area that clicks to chat to other players in the match.
D) Edit player : Button to edit player detail such as skin or name.
E) Match setting : Button that only the host can click to edit the role in the match.
F) **How to play button :** Button to show how the game is played or the rule used in the game briefly.



A) **Reveal role button :** Button to reveal player's role (not including other except it is a perk of his or her role).
B) **How to play button :** Button to show how the game is played or the rule used in the game briefly.
C) **Result :** Show the result of voting.

Only the team leader can see this button to confirm the team member.

Every player can see this button to vote whether they agree with the team leader or not.

Only team members who are not on the evil team can see this and the mission can only choose success.

Only team members who are on the evil team can see this and it is up to player to choose whether to let the mission fail or not.

Everybody will see this if the player vote is to reject the team member that is selected by the team leader. If the reject mark is full the evil will win the game.

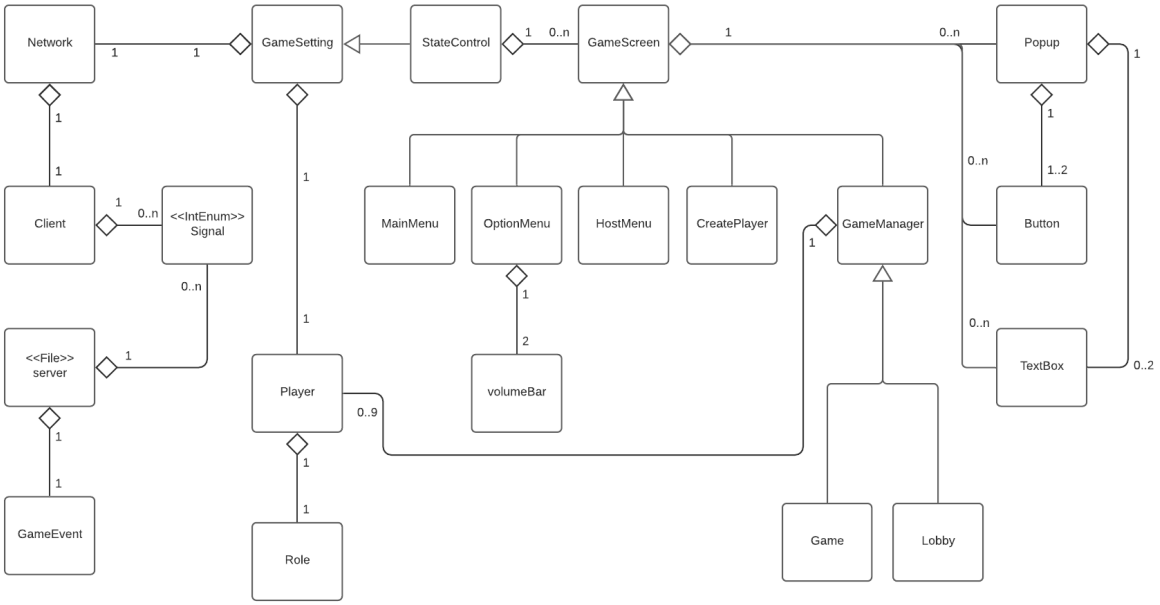
Everybody will see the green correct mark if the mission is successful.

Everybody will see the red cross mark if the mission is failed.

D) **Select party :** Only the team leader can see this to select the team members.

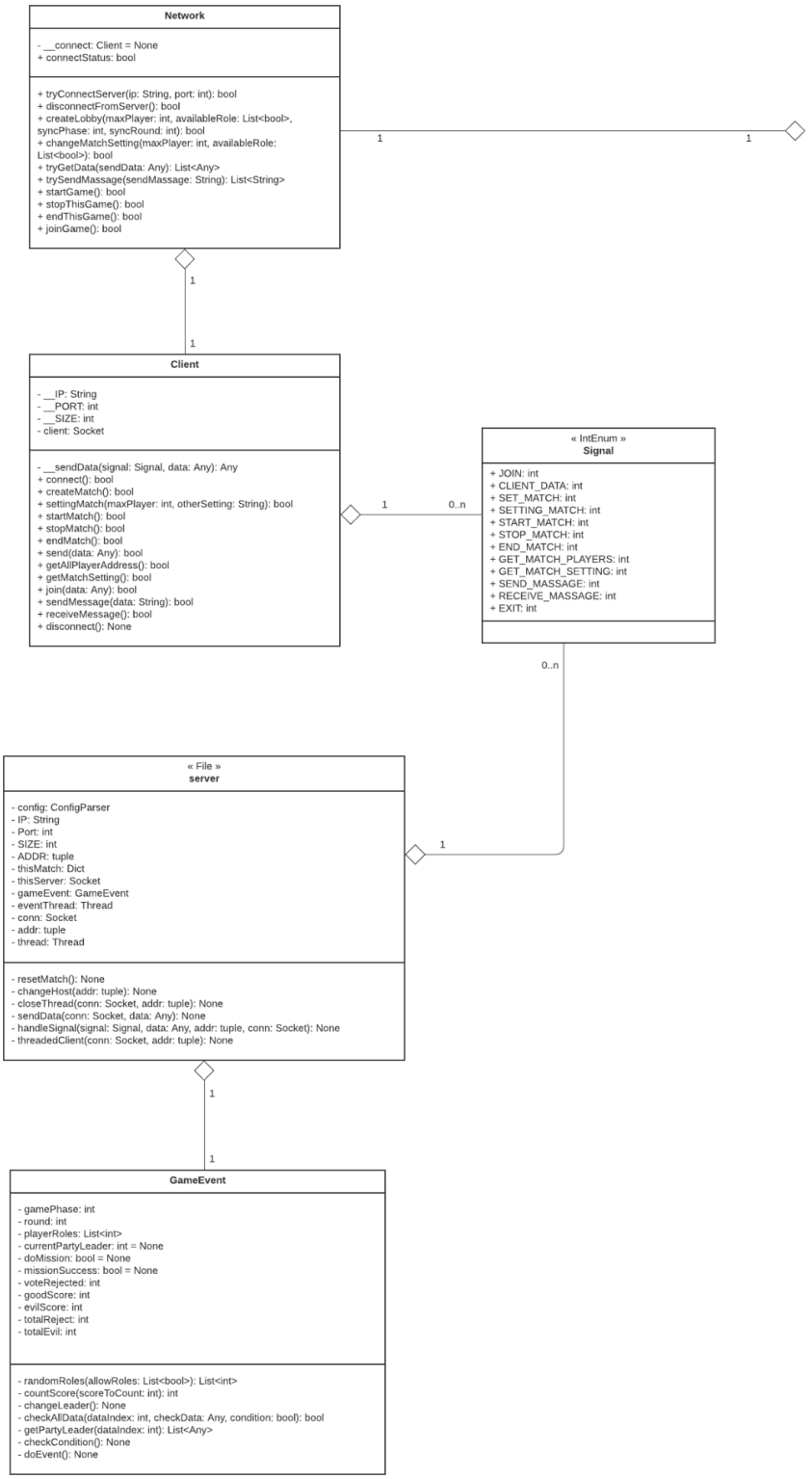
Class diagrams

Simplified Class Diagram

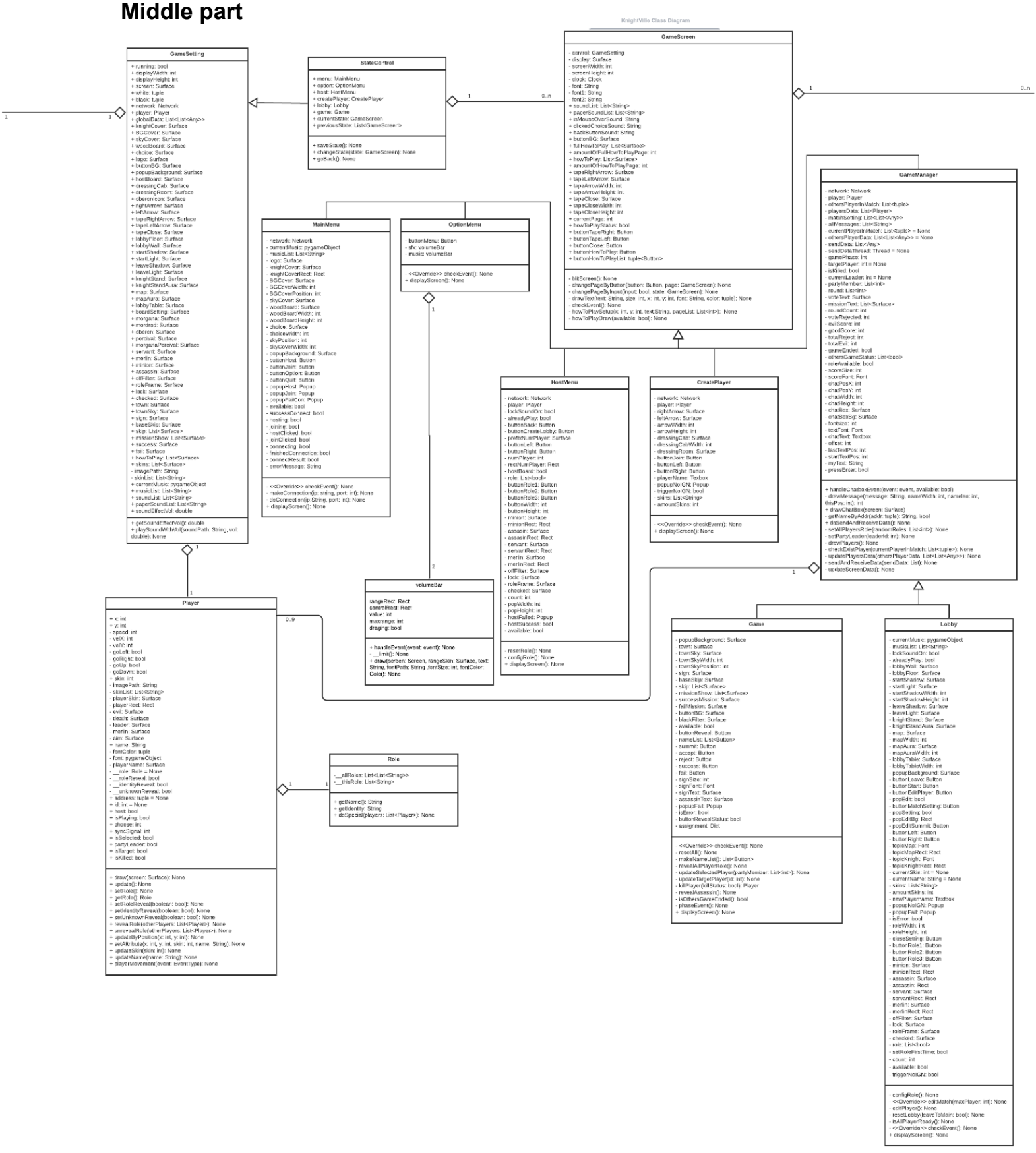


See full class diagram here:
https://lucid.app/lucidchart/08134123-244f-4da8-af7e-8639223c0684/edit?viewport_loc=-20%2C-67%2C1637%2C982%2CHWEp-vi-RSFO&invitationId=inv_f7b8cc16-4d06-41d9-83a2-71e0adea727f

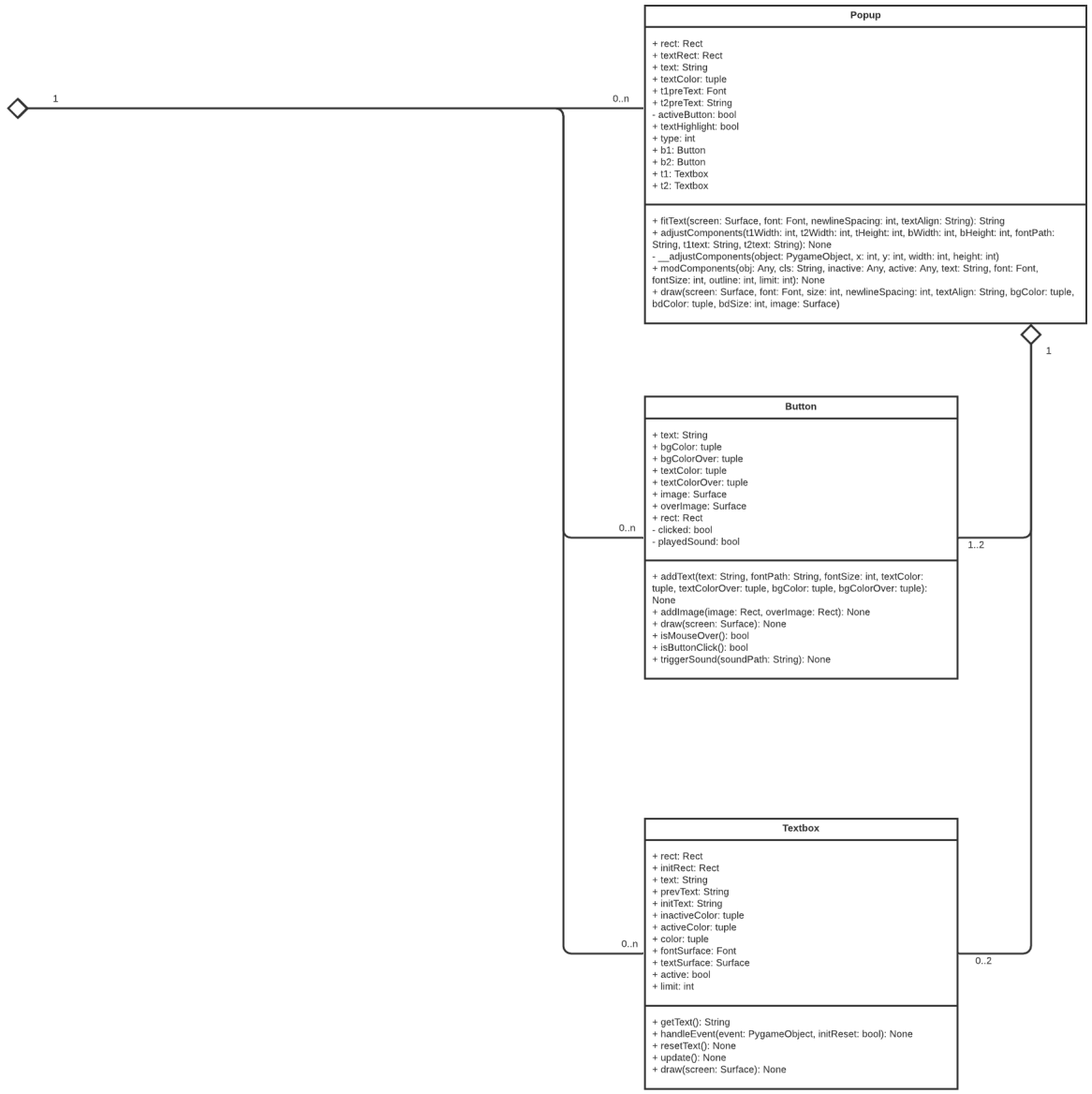
Left part


Network
- __connect: Client = None
+ connectStatus: bool
+ tryConnectServer(ip: String, port: int): bool
+ disconnectFromServer(): bool
+ createLobby(maxPlayer: int, availableRole: List<bool>, syncPhase: int, syncRound: int): bool
+ changeMatchSetting(maxPlayer: int, availableRole: List<bool>): bool
+ tryGetData(sendData: Any): List<Any>
+ trySendMassage(sendMassage: String): List<String>
+ startGame(): bool
+ stopThisGame(): bool
+ endThisGame(): bool
+ joinGame(): bool
1
1
1
1
Client
- __IP: String
- __PORT: int
- __SIZE: int
- client: Socket
- __sendData(signal: Signal, data: Any): Any
+ connect(): bool
+ createMatch(): bool
+ settingMatch(maxPlayer: int, otherSetting: String): bool
+ startMatch(): bool
+ stopMatch(): bool
+ endMatch(): bool
+ send(data: Any): bool
+ getAllPlayerAddress(): bool
+ getMatchSetting(): bool
+ join(data: Any): bool
+ sendMessage(data: String): bool
+ receiveMessage(): bool
+ disconnect(): None
1
0..n
« IntEnum »
Signal
+ JOIN: int
+ CLIENT_DATA: int
+ SET_MATCH: int
+ SETTING_MATCH: int
+ START_MATCH: int
+ STOP_MATCH: int
+ END_MATCH: int
+ GET_MATCH_PLAYERS: int
+ GET_MATCH_SETTING: int
+ SEND_MASSAGE: int
+ RECEIVE_MASSAGE: int
+ EXIT: int
0..n
1
« File »
server
- config: ConfigParser
- IP: String
- Port: int
- SIZE: int
- ADDR: tuple
- thisMatch: Dict
- thisServer: Socket
- gameEvent: GameEvent
- eventThread: Thread
- conn: Socket
- addr: tuple
- thread: Thread
- resetMatch(): None
- changeHost(addr: tuple): None
- closeThread(conn: Socket, addr: tuple): None
- sendData(conn: Socket, data: Any): None
- handleSignal(signal: Signal, data: Any, addr: tuple, conn: Socket): None
- threadedClient(conn: Socket, addr: tuple): None
1
1
GameEvent
- gamePhase: int
- round: int
- playerRoles: List<int>
- currentPartyLeader: int = None
- doMission: bool = None
- missionSuccess: bool = None
- voteRejected: int
- goodScore: int
- evilScore: int
- totalReject: int
- totalEvil: int
- randomRoles(allowRoles: List<bool>): List<int>
- countScore(scoreToCount: int): int
- changeLeader(): None
- checkAllData(dataIndex: int, checkData: Any, condition: bool): bool
- getPartyLeader(dataIndex: int): List<Any>
- checkCondition(): None
- doEvent(): None

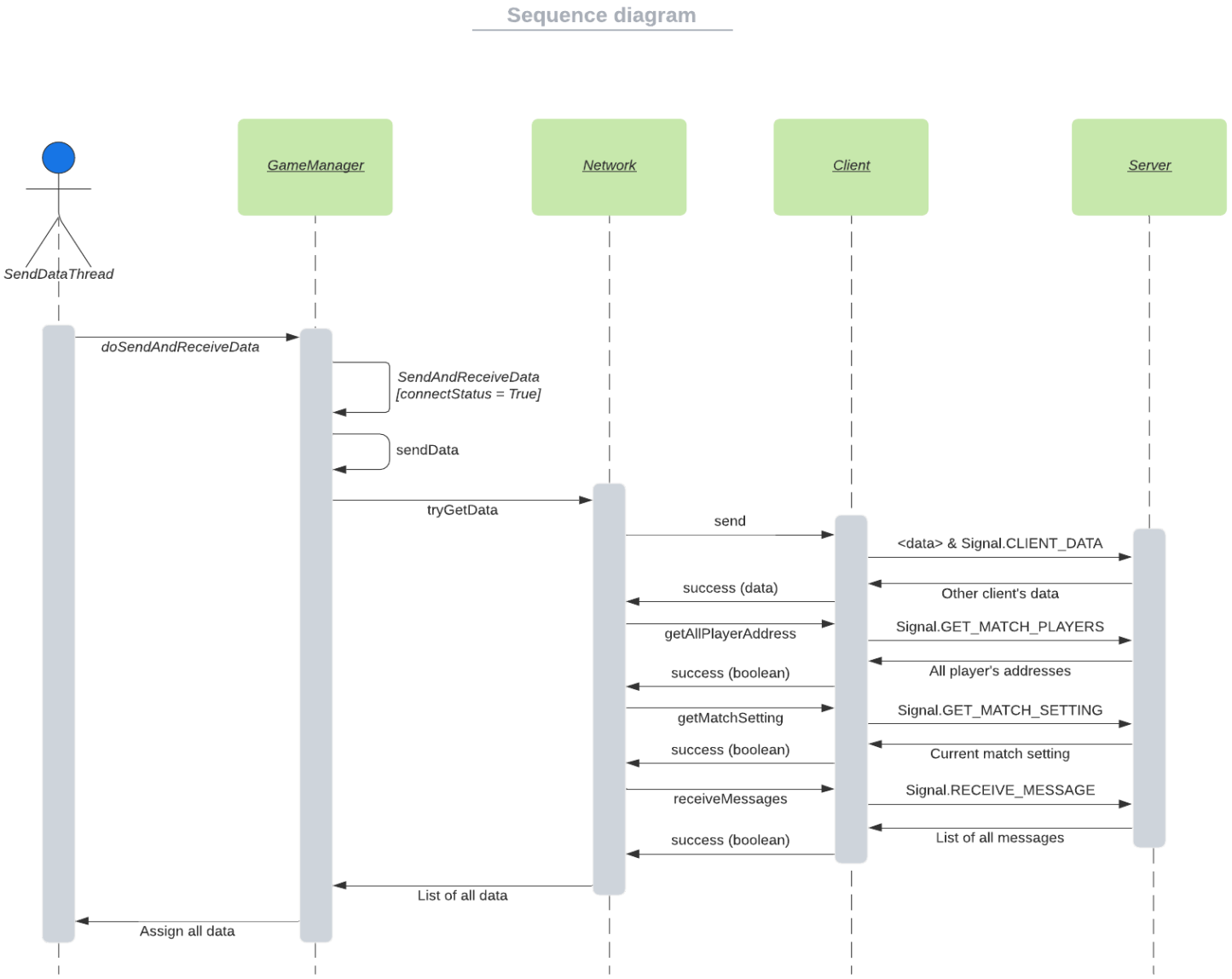
Middle part


KnightVille Class Diagram
GameSetting
+ running: bool
+ displayWidth: int
+ displayHeight: int
+ screen: Surface
+ white: tuple
+ black: tuple
+ network: Network
+ player: Player
+ globalData: List<List<Any>>
+ knightCover: Surface
+ BGCover: Surface
+ skyCover: Surface
+ woodBoard: Surface
+ choice: Surface
+ logo: Surface
+ buttonBG: Surface
+ popupBackground: Surface
+ hostBoard: Surface
+ dressingCab: Surface
+ dressingRoom: Surface
+ oberonIcon: Surface
+ rightArrow: Surface
+ leftArrow: Surface
+ tapeRightArrow: Surface
+ tapeLeftArrow: Surface
+ tapeClose: Surface
+ lobbyFloor: Surface
+ lobbyWall: Surface
+ startShadow: Surface
+ startLight: Surface
+ leaveShadow: Surface
+ leaveLight: Surface
+ knightStand: Surface
+ knightStandAura: Surface
+ map: Surface
+ mapAura: Surface
+ lobbyTable: Surface
+ boardSetting: Surface
+ morgana: Surface
+ mordred: Surface
+ oberon: Surface
+ percival: Surface
+ morganaPercival: Surface
+ servant: Surface
+ merlin: Surface
+ minion: Surface
+ assassin: Surface
+ offFilter: Surface
+ roleFrame: Surface
+ lock: Surface
+ checked: Surface
+ town: Surface
+ townSky: Surface
+ sign: Surface
+ baseSkip: Surface
+ skip: List<Surface>
+ missionShow: List<Surface>
+ success: Surface
+ fail: Surface
+ howToPlay: List<Surface>
+ skins: List<Surface>
- imagePath: String
- skinList: List<String>
+ currentMusic: pygameObject
+ musicList: List<String>
+ soundList: List<String>
+ paperSoundList: List<String>
+ soundEffectVol: double
+ getSoundEffectVol(): double
+ playSoundWithVol(soundPath: String, vol: double): None
StateControl
+ menu: MainMenu
+ option: OptionMenu
+ host: HostMenu
+ createPlayer: CreatePlayer
+ lobby: Lobby
+ game: Game
+ currentState: GameScreen
+ previousState: List<GameScreen>
+ saveState(): None
+ changeState(state: GameScreen): None
+ goBack(): None
GameScreen
- control: GameSetting
- display: Surface
- screenWidth: int
- screenHeight: int
- clock: Clock
- font: String
- font1: String
- font2: String
+ soundList: List<String>
+ paperSoundList: List<String>
+ isMouseOverSound: String
+ clickedChoiceSound: String
+ backButtonSound: String
+ buttonBG: Surface
+ fullHowToPlay: List<Surface>
+ amountOfFullHowToPlayPage: int
+ howToPlay: List<Surface>
+ amountOfHowToPlayPage: int
+ tapeRightArrow: Surface
+ tapeLeftArrow: Surface
+ tapeArrowWidth: int
+ tapeArrowHeight: int
+ tapeClose: Surface
+ tapeCloseWidth: int
+ tapeCloseHeight: int
+ currentPage: int
+ howToPlayStatus: bool
+ buttonTapeRight: Button
+ buttonTapeLeft: Button
+ buttonClose: Button
+ buttonHowToPlay: Button
+ buttonHowToPlayList: tuple<Button>
- blitScreen(): None
- changePageByButton(button: Button, page: GameScreen): None
- changePageByInput(input: bool, state: GameScreen): None
- drawText(text: String, size: int, x: int, y: int, font: String, color: tuple): None
- checkEvent(): None
- howToPlaySetup(x: int, y: int, text:String, pageList: List<int>): None
- howToPlayDraw(available: bool): None
GameManager
- network: Network
- player: Player
- othersPlayerInMatch: List<tuple>
- playersData: List<Player>
- matchSetting: List<List<Any>>
- allMessages: List<String>
- currentPlayerInMatch: List<tuple> = None
- othersPlayerData: List<List<Any>> = None
- sendData: List<Any>
- sendDataThread: Thread = None
- gamePhase: int
- targetPlayer: int = None
- isKilled: bool
- currentLeader: int = None
- partyMember: List<int>
- round: List<int>
- voteText: Surface
- missionText: List<Surface>
- roundCount: int
- voteRejected: int
- evilScore: int
- goodScore: int
- totalReject: int
- totalEvil: int
- gameEnded: bool
- othersGameStatus: List<bool>
- roleAvailable: bool
- scoreSize: int
- scoreFont: Font
- chatPosX: int
- chatPosY: int
- chatWidth: int
- chatHeight: int
- chatBox: Surface
- chatBoxBg: Surface
- fontsize: int
- textFont: Font
- chatText: Textbox
- offset: int
- lastTextPos: int
- startTextPos: int
- myText: String
- pressEnter: bool
+ handleChatboxEvent(even: event, available: bool)
- drawMessage(message: String, nameWidth: int, nameLen: int, thisPos: int): int
+ drawChatBox(screen: Surface)
- getNameByAddr(addr: tuple): String, bool
+ dbSendAndReceiveData(): None
- setAllPlayersRole(randomRoles: List<int>): None
- setPartyLeader(leaderId: int): None
- drawPlayers(): None
- checkExistPlayer(currentPlayerInMatch: List<tuple>): None
- updatePlayersData(othersPlayerData: List<List<Any>>): None
- sendAndReceiveData(sendData: List): None
- updateScreenData(): None
MainMenu
- network: Network
- currentMusic: pygameObject
- musicList: List<String>
- logo: Surface
- knightCover: Surface
- knightCoverRect: Rect
- BGCover: Surface
- BGCoverWidth: int
- BGCoverPosition: int
- skyCover: Surface
- woodBoard: Surface
- woodBoardWidth: int
- woodBoardHeight: int
- choice: Surface
- choiceWidth: int
- skyPosition: int
- skyCoverWidth: int
- popupBackground: Surface
- buttonHost: Button
- buttonJoin: Button
- buttonOption: Button
- buttonQuit: Button
- popupHost: Popup
- popupJoin: Popup
- popupFailCon: Popup
- available: bool
- successConnect: bool
- hosting: bool
- joining: bool
- hostClicked: bool
- joinClicked: bool
- connecting: bool
- finishedConnection: bool
- connectResult: bool
- errorMessage: String
- <<Override>> checkEvent(): None
- makeConnection(ip: string, port: int): None
- doConnection(ip:String, port: int): None
+ displayScreen(): None
OptionMenu
- buttonMenu: Button
- sfx: volumeBar
- music: volumeBar
- <<Override>> checkEvent(): None
+ displayScreen(): None
HostMenu
- network: Network
- player: Player
- lockSoundOn: bool
- alreadyPlay: bool
- buttonBack: Button
- buttonCreateLobby: Button
- prefixNumPlayer: Surface
- buttonLeft: Button
- buttonRight: Button
- numPlayer: int
- rectNumPlayer: Rect
- hostBoard: bool
- role: List<bool>
- buttonRole1: Button
- buttonRole2: Button
- buttonRole2: Button
- buttonWidth: int
- buttonHeight: int
- minion: Surface
- minionRect: Rect
- assasin: Surface
- assasinRect: Rect
- servant: Surface
- servantRect: Rect
- merlin: Surface
- merlinRect: Rect
- offFilter: Surface
- lock: Surface
- roleFrame: Surface
- checked: Surface
- count: int
- popWidth: int
- popHeight: int
- hostFailed: Popup
- hostSuccess: bool
- available: bool
- resetRole(): None
- configRole(): None
+ displayScreen(): None
CreatePlayer
- network: Network
- player: Player
- rightArrow: Surface
- leftArrow: Surface
- arrowWidth: int
- arrowHeight: int
- dressingCab: Surface
- dressingCabWidth: int
- dressingRoom: Surface
- buttonJoin: Button
- buttonLeft: Button
- buttonRight: Button
- playerName: Textbox
- popupNoIGN: Popup
- triggerNoIGN: bool
- skins: List<String>
- amountSkins: int
- <<Override>> checkEvent(): None
+ displayScreen(): None
volumeBar
rangeRect: Rect
controlRect: Rect
value: int
maxrange: int
draging: bool
+ handleEvent(event: event): None
- __init(): None
+ draw(screen: Screen, rangeSkin: Surface, text: String, frontPath: String ,frontSize: int, frontColor: Color): None
Player
+ x: int
+ y: int
- speed: int
- velX: int
- velY: int
- goLeft: bool
- goRight: bool
- goUp: bool
- goDown: bool
+ skin: int
- imagePath: String
- skinList: List<String>
- playerSkin: Surface
- playerRect: Rect
- evil: Surface
- clean: Surface
- leader: Surface
- merlin: Surface
- aim: Surface
+ name: String
- fontColor: tuple
- font: pygameObject
- playerName: Surface
- __role: Role = None
- __roleRevealed: bool
- __identityRevealed: bool
- __unknownRevealed: bool
+ address: tuple = None
+ id: int = None
+ host: bool
+ isPlaying: bool
+ choose: int
+ syncSignal: int
+ isSelected: bool
+ partyLeader: bool
+ isTarget: bool
+ isKilled: bool
+ draw(screen: Surface): None
+ update(): None
+ setRole(): None
+ getRole(): Role
+ setRoleRevealed(boolean: bool): None
+ setIdentityRevealed(boolean: bool): None
+ setUnknownRevealed(boolean: bool): None
+ revealRole(otherPlayers: List<Player>): None
+ unrevealRole(otherPlayers: List<Player>): None
+ updateByPosition(x: int, y: int): None
+ setAttribute(x: int, y: int, skin: int, name: String): None
+ updateSkin(skin: int): None
+ updateName(name: String): None
+ playerMovement(event: EventType): None
Role
- __allRoles: List<List<String>>
- __ifIsRole: List<String>
- getName(): String
+ getIdentity: String
- doSpecial(players: List<Player>): None
Game
- popupBackground: Surface
- town: Surface
- townSky: Surface
- townSkyWidth: int
- townSkyPosition: int
- sign: Surface
- baseSkip: Surface
- skip: List<Surface>
- missionShow: List<Surface>
- successMission: Surface
- failMission: Surface
- buttonBG: Surface
- blackFilter: Surface
- available: bool
- buttonReveal: Button
- nameList: List<Button>
- summit: Button
- accept: Button
- reject: Button
- success: Button
- fail: Button
- signText: Surface
- assassinText: Surface
- popupFail: Popup
- isError: bool
- buttonRevealStatus: bool
- assignment: Dict
- <<Override>> checkEvent(): None
- resetAll(): None
- makeNameList(): List<Button>
- revealAllPlayerRole(): None
- updateSelectedPlayer(partyMember: List<int>): None
- updateTargetPlayer(id: int): None
- killPlayer(killStatus: bool): Player
- revealAssassin(): None
- isOthersGameEnded(): bool
- phaseEvent(): None
+ displayScreen(): None
Lobby
- currentMusic: pygameObject
- musicList: List<String>
- lockSoundOn: bool
- alreadyPlay: bool
- lobbyWall: Surface
- lobbyFloor: Surface
- startShadow: Surface
- startLight: Surface
- startShadowWidth: int
- startShadowHeight: int
- leaveShadow: Surface
- leaveLight: Surface
- knightStand: Surface
- knightStandAura: Surface
- map: Surface
- mapWidth: int
- mapAura: Surface
- mapAuraWidth: int
- lobbyTable: Surface
- lobbyTableWidth: int
- popupBackground: Surface
- buttonLeave: Button
- buttonStart: Button
- buttonEditPlayer: Button
- popEdit: bool
- buttonMatchSetting: Button
- popSetting: bool
- popEditBg: Rect
- popEditSummit: Button
- buttonLeft: Button
- buttonRight: Button
- topicMap: Font
- topicMapRect: Rect
- topicKnight: Font
- topicKnightRect: Rect
- currentSkin: int = None
- currentName: String = None
- skins: List<String>
- amountSkins: int
- newPlayername: Textbox
- popupNoIGN: Popup
- popupFail: Popup
- isError: bool
- roleWidth: int
- roleHeight: int
- closeSetting: Button
- buttonRole1: Button
- buttonRole2: Button
- buttonRole3: Button
- minion: Surface
- minionRect: Rect
- assassin: Surface
- assassin: Rect
- servant: Surface
- servantRect: Rect
- merlin: Surface
- merlinRect: Rect
- offFilter: Surface
- lock: Surface
- roleFrame: Surface
- checked: Surface
- role: List<bool>
- setRoleFirstTime: bool
- count: int
- available: bool
- triggerNoIGN: bool
- configRole(): None
- <<Override>> editMatch(maxPlayer: int): None
- editPlayer(): None
- resetLobbyJleaveToMain: bool): None
- isAllPlayerReady(): None
- <<Override>> checkEvent(): None
+ displayScreen(): None

Right Part


Popup
+ rect: Rect
+ textRect: Rect
+ text: String
+ textColor: tuple
+ t1preText: Font
+ t2preText: String
- activeButton: bool
+ textHighlight: bool
+ type: int
+ b1: Button
+ b2: Button
+ t1: Textbox
+ t2: Textbox
+ fitText(screen: Surface, font: Font, newlineSpacing: int, textAlign: String): String
+ adjustComponents(t1Width: int, t2Width: int, tHeight: int, bWidth: int, bHeight: int, fontPath: String, t1text: String, t2text: String): None
- __adjustComponents(object: PygameObject, x: int, y: int, width: int, height: int)
+ modComponents(obj: Any, cls: String, inactive: Any, active: Any, text: String, font: Font, fontSize: int, outline: int, limit: int): None
+ draw(screen: Surface, font: Font, size: int, newlineSpacing: int, textAlign: String, bgColor: tuple, bdColor: tuple, bdSize: int, image: Surface)
1
0..n
1
Button
+ text: String
+ bgColor: tuple
+ bgColorOver: tuple
+ textColor: tuple
+ textColorOver: tuple
+ image: Surface
+ overImage: Surface
+ rect: Rect
- clicked: bool
- playedSound: bool
+ addText(text: String, fontPath: String, fontSize: int, textColor: tuple, textColorOver: tuple, bgColor: tuple, bgColorOver: tuple): None
+ addImage(image: Rect, overImage: Rect): None
+ draw(screen: Surface): None
+ isMouseOver(): bool
+ isButtonClick(): bool
+ triggerSound(soundPath: String): None
0..n
1..2
Textbox
+ rect: Rect
+ initRect: Rect
+ text: String
+ prevText: String
+ initText: String
+ inactiveColor: tuple
+ activeColor: tuple
+ color: tuple
+ fontSurface: Font
+ textSurface: Surface
+ active: bool
+ limit: int
+ getText(): String
+ handleEvent(event: PygameObject, initReset: bool): None
+ resetText(): None
+ update(): None
+ draw(screen: Surface): None
0..n
0..2

Sending and Receiving Data Method



This sequence diagram shows how “SendDataThread” works in Lobby and Match. This thread will run immediately after the Lobby and Match scene are drawn. Then, the thread will call the method “doSendAndReceiveData” in GameManager, this method responsible for checking connection status and getting “sendData” (data to be sent). Content inside “sendData” will be different between Lobby and Match.

If connection status is True and sendData are prepared, “doSendAndReceiveData” will call “tryGetData” which is a method in Network responsible for getting any necessary data for our game. This method will call another four method from Client which are
“send” - Send our own data with Signal.CLIENT_DATA to the Server and Server will provide data of other clients (players) back.

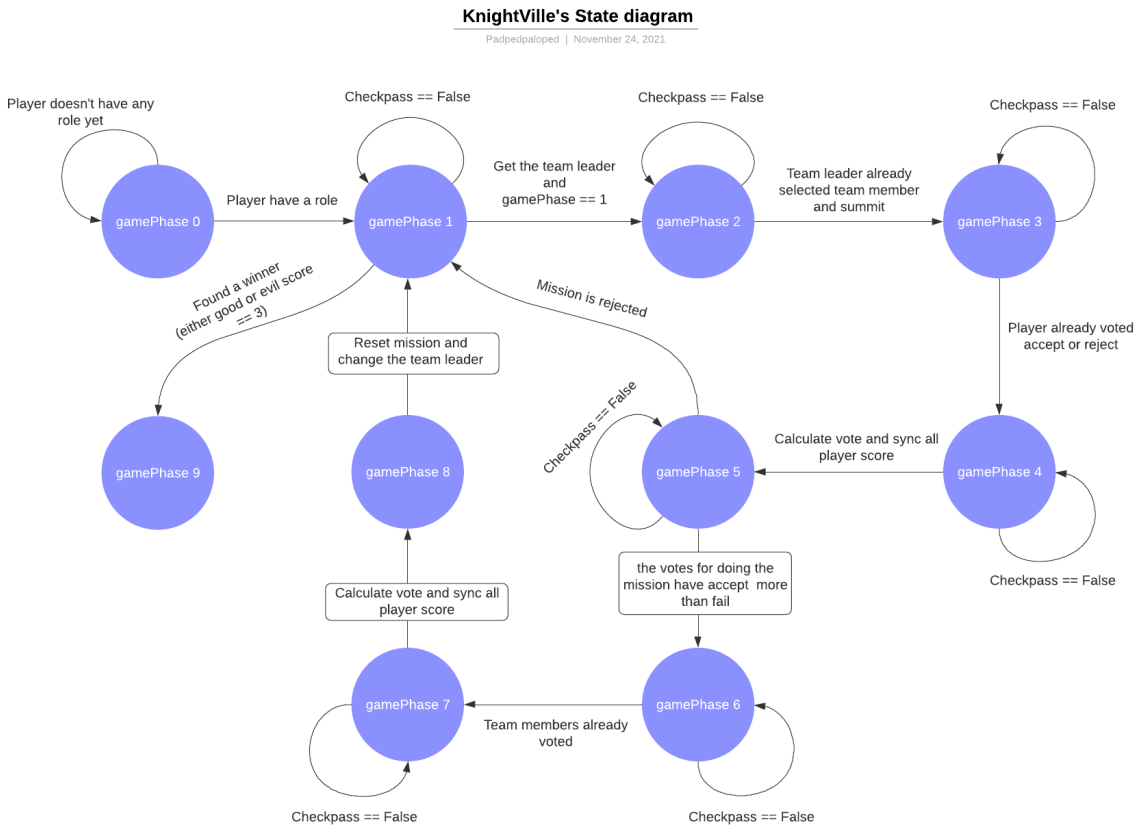
“getAllPlayerAddress” - Send Signal.GET_MATCH_PLAYERS to the Server then Server will respond back with all client’s addresses (including your address).

“getMatchSetting” - Send Signal.GET_MATCH_SETTING to the Server then Server will respond back with the current match setting.

“receiveMessages” - Send Signal.RECEIVE_MESSAGE to the Server and Server will provide all messages of every player back (including your message). (all message will be shown in chat box)

In-game state explanation

State diagram below explain how in game (match) works in each phase



Note: *checkPass* is the variable to indicate whether to move to the next state or not. If the *checkPass* is *True*, the state machine will be moved to the next state(gamePhase) but, if *checkPass* is *False*, then stay at the same state.

gamePhase 0:

1.) Role randomization.

gamePhase 1:

1.) Reset all variables which were used in previous mission, reset previous team's members and previous selected player status.
2.) Check the score of each team, if we have the winner then go to **gamePhase 9**.
3.) Get the team leader and check that everyone has the same team leader (data is already synchronized) and then change the *checkPass* to *True*.

gamePhase 2:

1.) Assign the number of players in the current mission.
2.) Make a list of the player's name to be selected by the team leader.
3.) Make a loop to create a button that contains the player's name for the team leader, to select the current team members.
4.) Check the team leader's choice and team member for every player is the same (data is synchronized), then give *checkPass* to *True*.

gamePhase 3:

1.) Create an accept and reject button and check the input from the team members.

gamePhase 4:

1.) Calculate the vote from each player and synchronize the score.
2.) Give *checkPass* to *True*.

gamePhase 5:

1.) Check if the votes for the current mission are accept more than fail, then Give *checkPass* to *True* to go to the next state(**gamePhase6**).
2.) If the current mission is rejected, change the team leader and then, go back to **gamePhase1**.

gamePhase 6:

1.) Create the button for team members to make a vote.
2.) Get the votes from team members.
3.) Give *checkPass* to *True*.

gamePhase 7:

1.) Calculate the votes from team members(success or fail) and synchronize all player scores.
2.) Give *checkPass* to *True*.

gamePhase 8:

1.) Reset the mission, change the team leader and go to **gamePhase1.**

gamePhase 9:

1.) Conclude the match.

Art of KnightVille

Inspiration

In our game, we retheme Avalon to be a knight team and a demon theme. And because this game we created is based on Avalon, we want to keep the concept that no matter what age or gender you are, you can still enjoy playing this game.

Before we get into the UI, character, logo, and map design, we want to explain why our game came out to be 2.5D perspective and in a paper-like art. The original idea comes from the tales that use paper as a character to tell a story (as in the picture below). It gives a feeling of childish and vivid vibe.

Example of story-telling by paper (นิทานกระดาษ)

Character design

At first, we designed characters as we could think of.

First design of characters

But in the end, we get all the characters for each role. Actually, we also created the characters that are not related to the role because all of these will be used as skins only, not used in the actual role in the game.

Loyal servant

Merlin

Percival

Assassin

Minion of Mordred

Mordred

Morgana

Oberon

And as you can see from the “What is Avalon?” In part, when explaining the roles in Avalon, we redesign the characters to be more suitable for the concept “Knight and Demon”.

Loyal Servant - as the game concept is Knight so we design Loyal servant to be a knight.
Merlin - this role has a special ability to see who is evil so we design this role like a fortune teller.
Percival - Percival is Merlin’s disciple. By this, I think Percival should give a child vibe.
Assassin - we make an Assassin as an angel of death because, at the end of a game, an Assassin can take one’s life. By this, we think Angel of death is suitable for this role.
Minion of Mordred - we use an adult version of Oberon.
Mordred - Mordred is the head of the Evil team so we design him to be a Demon lord.
Morgana - this role will appear as Merlin in the perspective of Percival so, as a demon, we think Morgana seems like a doppelganger.
Oberon - as Oberon is an Evil who does not really know many things so, in our game, Oberon turns out to be a baby demon.
Here are some other characters players can choose as skin in the game.

Evil Sheep

Pajamas Banana

Banana Gangster

Knight No.2

Old Merlin

One-eye Devil

Spider Demon

Logo

The next section is Logo, at first we wanted it to be colorful but when we combined it with a Menu UI in our game it did not fit well. So we recolor it and also make it to be an actual straight rectangle.

First draft

Second draft

Several color ideas of the logo

Here is the most important part, which is the meaning behind the logo. Our logo not only can be read as KnightVIIIe but it also can be read as Eight(E)VILs, if we ignore ‘n’ in knight and see ‘t’ in knight as ‘t’ and ‘E’ in the same time. (this is the reason why n’s color is different from K_ight). Furthermore, knight“VIII”e, “VIII " stands for 8 from roman numerals.

KnightVILLe

EightEvils

The reason why there are 8 is important for our game because there are 8 available roles in the game. In the end, we get the final version of the logo. (as in picture below)

The final version of the logo

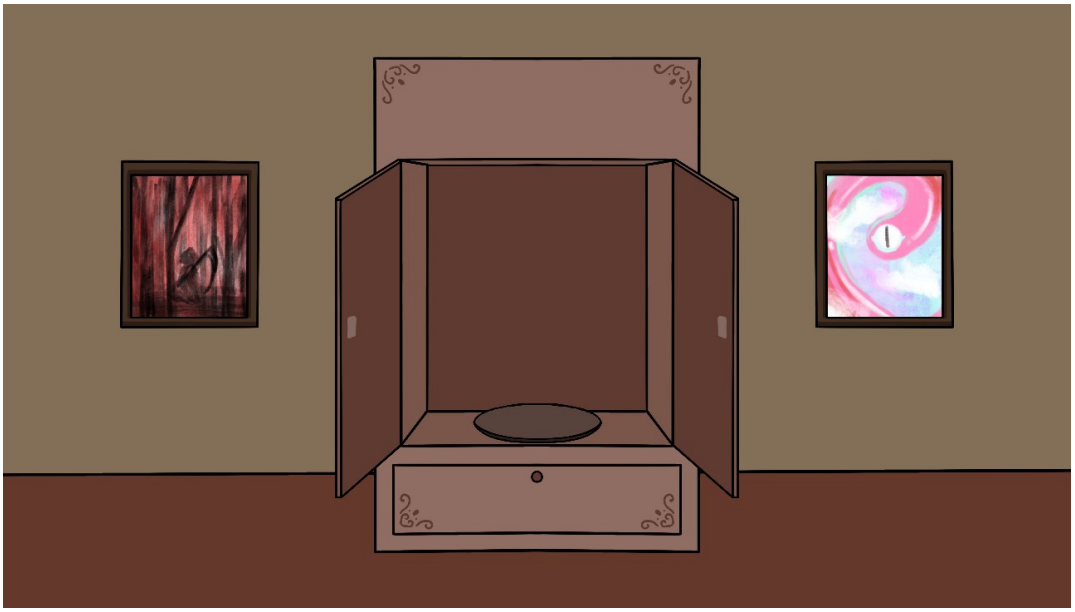
Menu & Lobby design

This is the menu design. The background is the Knight village town. And in front is wood and a knight, to bring it more outstanding.

Sketch work

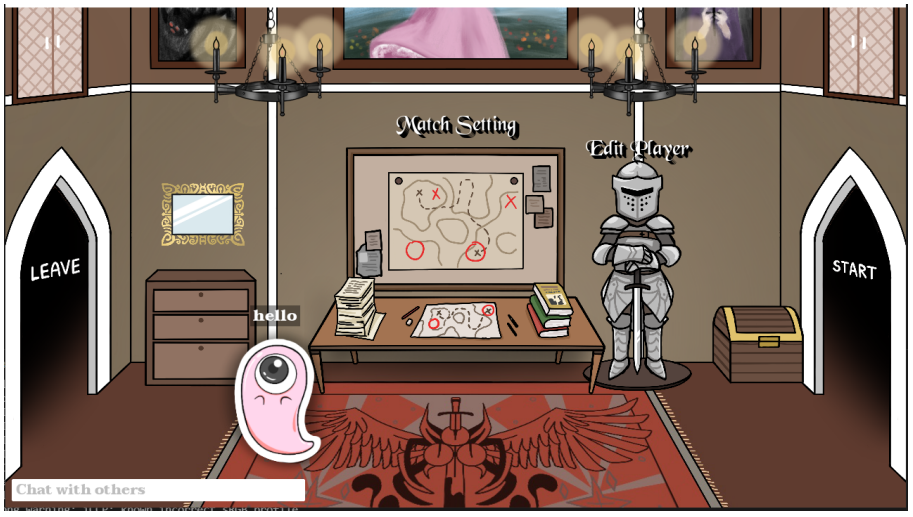
Real work

Before getting into the lobby, we need a page for changing skin and in-game name. By this, the idea is the dressing room. We also add a picture of Minion of Mordred and Assassin to tell that the knights once have faced these Evils and try to find out while doing the mission.



Changing skin and in-game name

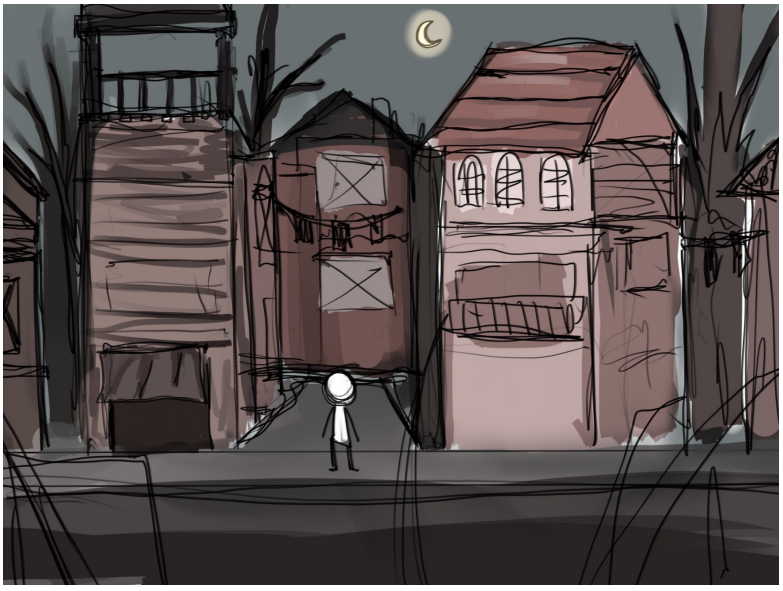
In the lobby part, we want the feeling to be like all players are in the meeting, discussing the plan for doing the missions and finding out who is EVIL. So we design it to be a room where everyone can gather before the game starts. The pattern of carpet on the floor is a sword stab through an eye of a demon, which seems like a spider, telling that at the end the demon will die and victory will be knights'. The spreaded wings on carpet stands for the glorious victory for the Villagers and knights. The pictures on the wall are Spider demon, Girl in dress, and Merlin. The spider demon picture is to remind everybody in the village of the appearance of the demons. The Merlin picture was painted by one of the villagers, who believe that Merlin will be the one that brings hope back to this village. The middle picture, "Girl in dress", is the first Merlin in this village, she can point out the demon in disguise. She is the only Merlin who can see the true identity of the Demon Lord, Mordred. Sadly, she was killed by the demon. After this tragedy happened, the demon seems to be more attracted to this village than it ever was.



Lobby

Game map design

The map in a game match, we want it to be gloomy, dark and give a medieval-era, witch hunt era atmosphere, this is our first draft. However, this draft is designed before the paper-like idea so it looks more dark and creepy.



The first draft

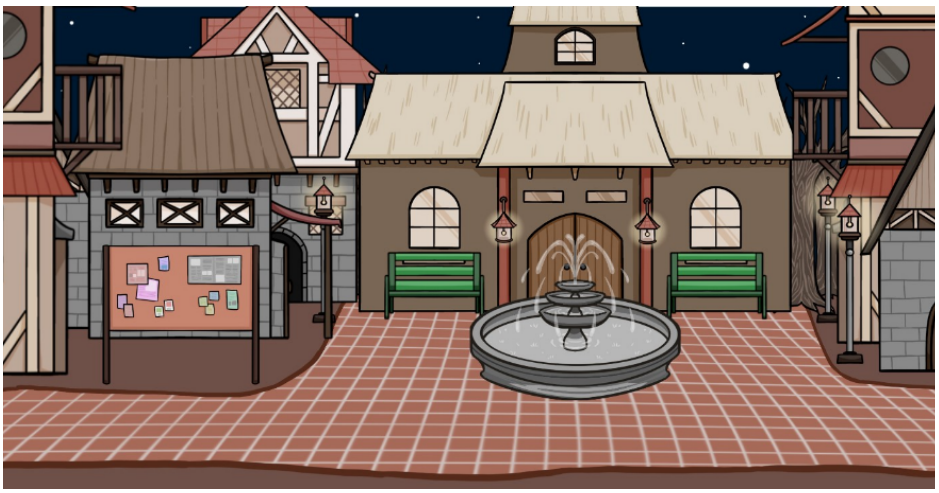
Then our second draft came out. This one is more paper-like and vivid, compared to the first draft. We add white strokes for every component and add the shadow behind them. The result from the draft is quite satisfying so we decided to make it this way. This gives a scent of childish, playful, and also a feeling of paper theater, which is great and interesting.

The second draft

The next step is to create an actual object for use in our game. At first, we sketch the buildings and some character skin that will be used in the game, to see if it fits or not.

Town sketch

Finally, we get the final version of the map. We cut out the stroke for all objects on the map because it will make the characters less attractive. The stroke on buildings will make the building stand out too much.



Map used in game

This is the full version of map, for further development for the game. The water foundation in the middle of the map is a place where all villagers and knights will gather up. Left side of the water foundation is a “town board”, where anyone can post or pin the missions or news for everyone to see. The right of the map is the exit of the town. It will lead to the “dark forest” that is full of danger and demons.

Full map design

Game Audio

For the game music and audio part, we mainly use FI studio for the main song *(Journey of the apprentice knight)* and background music *(The celtic village)* production. We design the main song and BGM to be similar to Tavern song in order to achieve the feeling of medieval traditional folk song. Two main elements that make the songs have the feeling of traditional music are the instruments used and the modalogy in music theory. The idea and the music theory used in the composition will be explained in the following part, so we can have more specification in detail.

Instruments and Music theory

Since all songs are produced in DAW(Digital audio workstation), all of the instruments that we use in production are VST plug-ins; not the real instrument. Ivory Wind (embertone.com) is the free instrument library that is chosen to be the main motif of the first part; it provides the sound of an 80 years-old wooden instrument. Another instrument is the Dulcimer from DSK World StringZ and LABS — spitfireaudio that contains the identity and intuition of an old traditional song which is suitable for our purpose. During the information gathering process we found that when composing music with Dorian(II) and Myxolydian(V) modes can provide some feeling of adventure for the old time traveller. When combined the theory with traditional instruments, the middle-age style drum and. other wooden instruments their identities came out explicitly; in our case, we use one from LABS — Drums.

The Ideas behind

We want to explain more about the main song which is ***Journey of the apprentice knight***. The composition idea is about the one who just became the knight, taking more and more practice. One day he was assigned the big tough job which is conveyed by many instruments playing together around the middle of the song. After he overcame that hard assignment, he became the great knight in the final part. So, you can identify the bigger and bigger scale in the song that we try to convey the story of the one apprentice knight.

For the sound effects that are played when the user clicks some button, we mostly selected the sound effect to comply with the UI design, in order to provide the feeling of controllability. Most sound effects come from the internet but, we need to equalize all sound effects and cut off some unimportant part before embedding it into pygame.

Development process description

Development process of our team was highly inspired from **Scrum** since it's more suitable for a little team and friendly to the developer. So, some adjustments are done to the original Scrum process to make it more appropriate to our styles and situation.
All changes that we have done are stated in the table below.

Scrum Workflow	Original Description	Our Workflow
Sprint	Work period (one month or less)	This we use the same as the original. Our work period is 1 week.
Sprint planning	Event at the beginning of a Sprint: - Agree on the sprint goal - Select product backlog - etc.	[Every Saturday 21:30 p.m. - 23:30 p.m.] (Maximum 2 hours) Event at the beginning of a week (Same day as the last day of the week.): - Select work to be done this week and assign it to each member. (At least 1 work per person.) - We use Kanban style as a product backlog. (We select our product backlog once and use the same for every sprint.) - We didn't assign Story points to each work. Priority of each work is up to the expertise of each team member.
Daily scrum	Daily meetings. (15 minutes or less)	We use periodic meetings instead of daily meetings. It can be any day at any time in a work period (Minimum of 3 times per week.) and each meeting took around 1 - 2 hours. (Content of each meeting depends on the situation at that time.)
Sprint review	Meeting at the end of a Sprint. (max 4 hours long)	[Every Saturday 21:30 p.m. - 23:30 p.m.] (Maximum 2 hours) We use the same day as a Planning day. - Update each person's work. (Current work status, Struggles, Bugs, etc.) - Update product backlog. - Go to Sprint planning. (Plan for next week)
Sprint retrospective	Meeting after Sprint review.	We have no activity similar to the Sprint retrospective.

Kanban - workflow management method for defining, managing and improving services that deliver knowledge work.
Story points - units of measure for expressing an estimate of the overall effort required to fully implement.

Reference:

- CPE 327 Lecture 2 - Software Process Frameworks
- https://en.wikipedia.org/wiki/Scrum_(software_development)
- https://www.atlassian.com/agile/project-management/estimation
- https://kanbanize.com/kanban-resources/getting-started/what-is-kanban

Summary of our workflow (from table)

- Work period is 1 week.
- Meeting every Saturday 21:30 p.m. - 23:30 p.m. (Maximum 2 hours)
  - Update work from previous week.
  - Plan work for next week : Use Kanban for product backlog, Each team member gets at least 1 work.
- Periodic meeting across the week. Meeting's content depends on the current situation and the number of meetings is 3 times minimum.

Roles

We have only 3 main roles as described in the table below.

Role's name	Duty
Project manager	- Make sure that projects follow the timeline. - Prioritize work and assign proper work to each team member. - Keep track of work and product backlogs. - Do some development tasks.
Process master (Scrum master - similar)	- Make sure that everyone can follow the workflow.
Developer	- Do assigned tasks. - Makes all decisions about the process. - Self-organizing. - Follow the workflow. **Sub roles** : Depending on expertise of each team member. Game Component Responsible for creating each component in the game for example button, popup, textbox, etc. User Interface Responsible for creating and arranging every component related to what users see for example scene's interface, icon, background, player, etc. Music Component Responsible for background music and sound effects. Tester Create, record and implement the test.

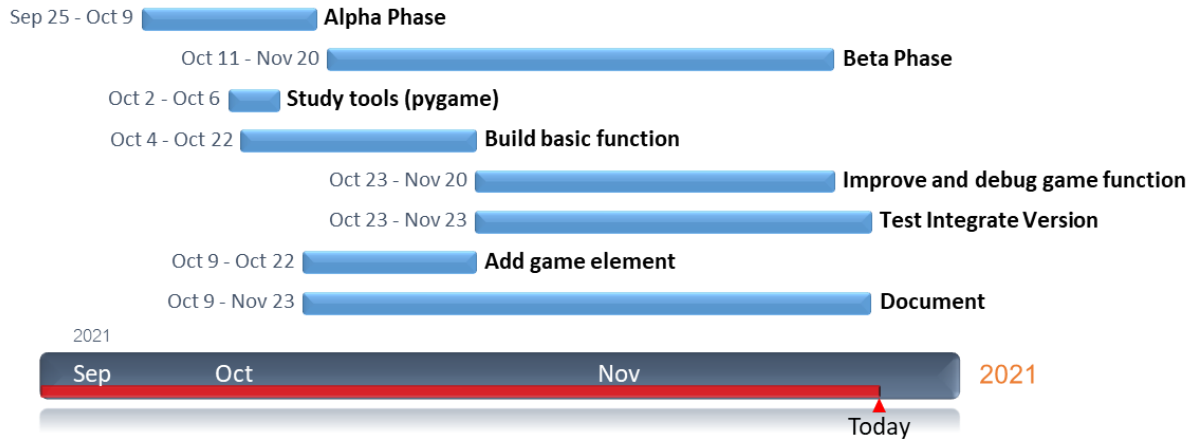
Team member and role

Name	ID	Role
Nithi Piyaphonnarinthon	62070503430	**Developer** (Tester)
Romtam Tanpituckpong	62070503444	**Developer** (Game Component)
Wagee Jr. Nanta Aree	62070503445	**Developer** (User Interface)
Chatchapon Sukitporn-udom	62070503455	**Project manager, Process master**
Kantawit Chatdamrongmongkol	62070503457	**Developer** (Game and Music Component)

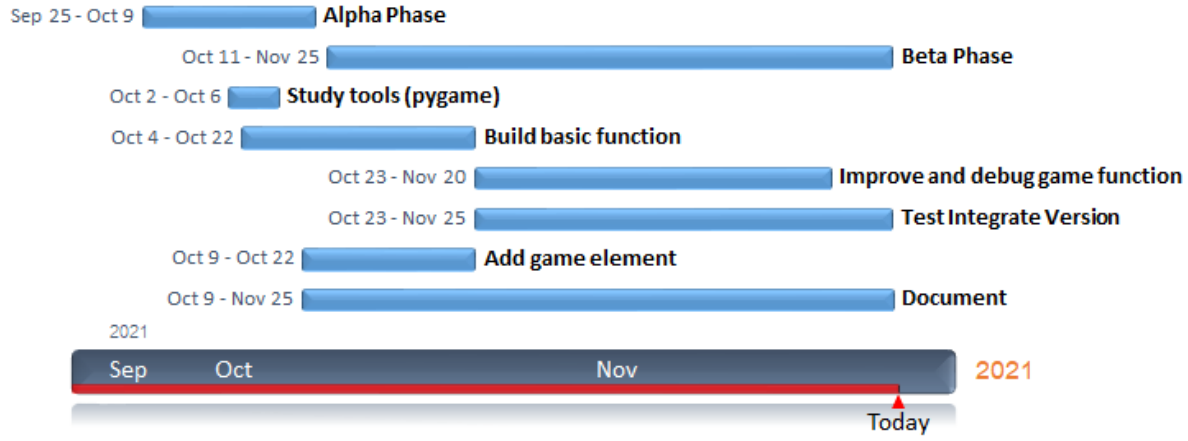
Project timeline

Week	Predetermine Goal
Week 1 (25 Sep 2021)	- Rule of working together. - Functionality and constraint.
Week 2 (2 Oct 2021)	- Use case diagram. - Use case narratives. - Learn development tools.
Week 3 (9 Oct 2021)	[Individual work] - Game state, rules, role. (Concept) - Basic network. - UI mockup. - Player.
Week 4 (16 Oct 2021)	[Integrate work] - All game scene structure. - Game logic already implemented. (state, rules, role) - Component diagram.
Week 5 (23 Oct 2021)	[Integrate work] - Game first version. (playable with core function / no proper UI) - Sequence diagram.
Week 6 (6 Nov 2021)	[Integrate work] - Optimize the game. - Add more UI. - Refactoring code. - Fixing bugs.
Week 7 (13 Nov 2021)	[Integrate work] - Game final version. (playable with all proper function and UI)
Week 8 (20 Nov 2021)	[Integrate work] - Test and fix bugs.

Expected timeline

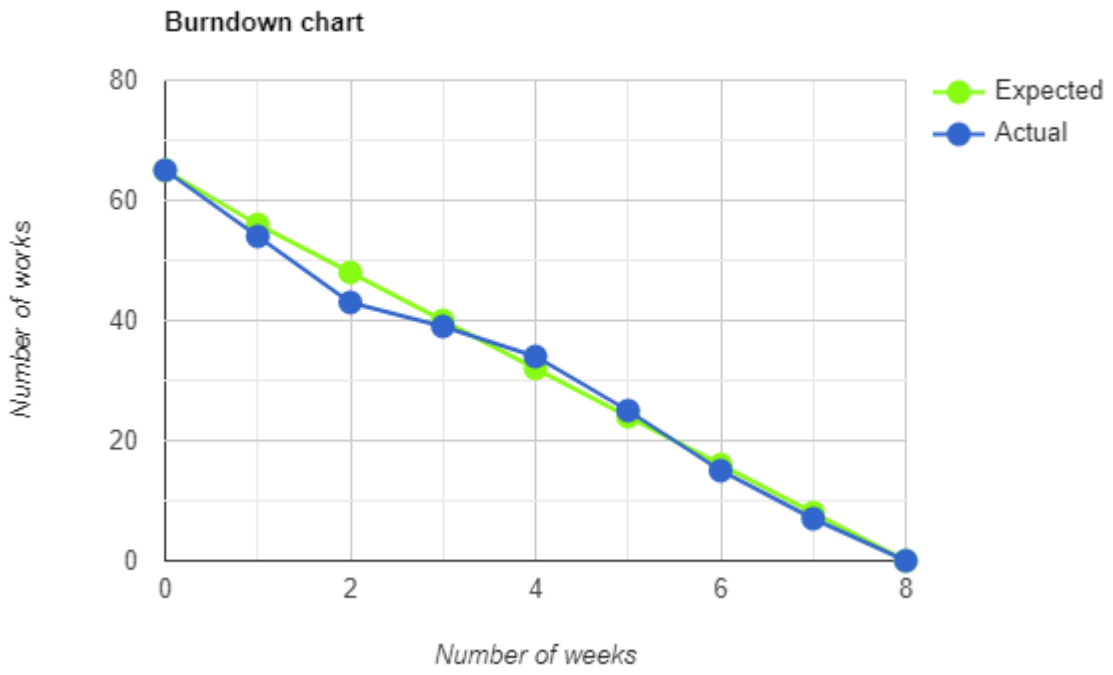


Actual timeline



Alpha Phase : Phase that everyone does the individual job according to their assignment and plans or their responsibility.
Beta Phase : Phase that integrates the basic function and component to build a game version.
Study tools : Study to implement the pygame library and try to build their own to learn how pygame works.
Build basic function : Build the function to create a necessary component to use in the game.
Improve and debug game function : Improve/Refactoring function and fix bug that occur in the test session.
Test Integrate Version : Test and record the bug detail such as when it occurred and analyze it to be used in the fixing session.
Add game element : Import element to project such as jpeg-png file and sound.
Document : Documenting that keeps track of the project.

Burndown chart



Since we don't have Story points like Scrum. So, each work is equal to 1 point.
We have a total of 65 works.

Result	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8
Expected	9	8	8	8	8	8	8	8
Actual	11	11	4	5	9	10	8	7

Note: Number in each cell is the number of works that should be finished in that week.

More info about each work in our Mirro:
https://miro.com/welcomeonboard/VUg5c01haEZWdGRTNWRsOVNkVlVNUzZMa2pwcEZvd3RnRk9ldDZZYk9yeG9CaGNXdHZIYjQ3cWJKaG84Y2N0V3wzMDc0NDU3MzU0MTg1MDUyNTE1?invite_link_id=977937242475

Tools

Software configuration management (Version Control)	**Github**	More info: https://github.com
	Github Desktop	More info: https://desktop.github.com
	Git	More info: https://git-scm.com
Integrated development environments (IDE)	**Visual Studio Code**	More info: https://code.visualstudio.com
Virtual Local Network (For online testing propose)	**Hamachi**	More info: https://www.vpn.net
Drawing software (All graphical elements)	**Procreate**	More info: https://procreate.art
Digital Audio Workstation (Producing Music)	**FI Studio 20**	More info: https://www.image-line.com
UI mockup	**Canva**	More info: https://www.canva.com
Online Whiteboard Tool (Backlog)	**Miro**	More info: https://www.miro.com
Diagram	**Lucidchart**	More info: https://www.lucidchart.com
	Visual Paradigm	More info: https://www.visual-paradigm.com
	Draw.IO	More info: https://www.draw.io

Programming language and libraries

Programming language	 Python	Version 3.9.8 More info: https://www.python.org/downloads/release/python-398
Libraries	**Non standard**	
	pygame 2.0.1	More info: https://www.pygame.org/news
	Standard	
	socket	More info: https://docs.python.org/3/library/socket.html
	threading	More info: https://docs.python.org/3/library/threading.html
	pickle	More info: https://docs.python.org/3/library/pickle.html
	random	More info: https://docs.python.org/3/library/random.html
	time	More info: https://docs.python.org/3/library/time.html
	configparser	More info: https://docs.python.org/3/library/configparser.html
	os	More info: https://docs.python.org/3/library/os.html
	sys	More info: https://docs.python.org/3/library/sys.html

Coding standard

Name - Camel case naming convention	**Example**
File	player.py createPlayer.py
Instance	name = “adam” thisCount = 0 doSomething = False

Method	def thisMethod(thatCount, thisNumber):
	  if thatCount > thisNumber:
	  return thatCount + thisNumber
	  else:
	  return thatCount

Name - Pascal case naming convention	**Example**

Class	class ThisClass:
	  def __init__(self):
	  pass

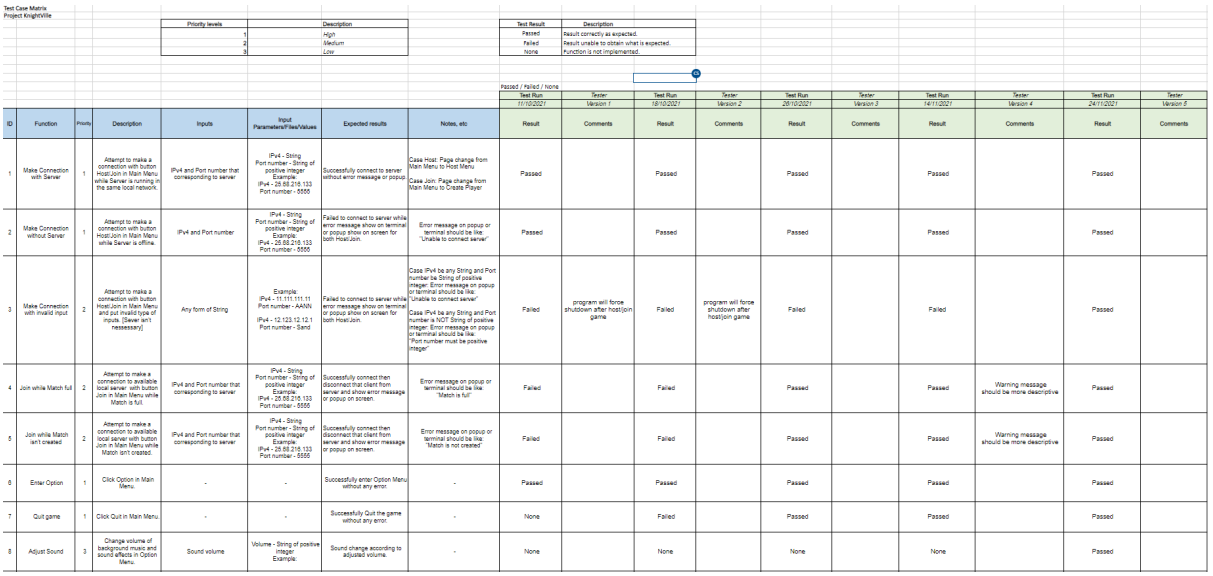
Comment	**Example** **Note:** [Description] is an explanation of that element and it’s free to design.
At the beginning of the File	‘’’ fileName - [Description] Last updated: 28 Oct 2021 ‘’’
Below Class’s name	‘’’ ClassName - [Description] ‘’’

Below Method’s name	‘’’
	methodName - [Description]
	+ parameter1 - [Description]
	+ parameter2 - [Description]
	
	+ return - [Description]
	  - <Value> if [Description]
	  - <Value> if [Description]
	‘’’

Comment inside method / class / file (Focus just about complicated task)	# [Description]

Bug testing

We are testing bugs in our game using Excel as you can see in the picture down below.



Test Case Matrix
Project KnightVille

Priority levels	Description
1	High
2	Medium
3	Low

Test Result	Description
Passed	Result correctly as expected.
Failed	Result unable to obtain what is expected.
None	Function is not implemented.

Passed / Failed / None

ID	Function	Priority	Description	Inputs	Input Parameters/Files/Values	Expected results	Notes, etc	Test Run 11/10/2021	Tester Version 1	Test Run 18/10/2021	Tester Version 2	Test Run 28/10/2021	Tester Version 3	Test Run 14/11/2021	Tester Version 4	Test Run 24/11/2021	Tester Version 5
								Result	Comments	Result	Comments	Result	Comments	Result	Comments	Result	Comments
1	Make Connection with Server	1	Attempt to make a connection with button Host/Join in Main Menu while Server is running in the same local network.	IPv4 and Port number that corresponding to server	IPv4 - String Port number - String of positive integer Example: IPv4 - 26.66.216.133 Port number - 5555	Successfully connect to server without error message or popup.	Case Host: Page change from Main Menu to Host Menu Case Join: Page change from Main Menu to Create Player	Passed		Passed		Passed		Passed		Passed	
2	Make Connection without Server	1	Attempt to make a connection with button Host/Join in Main Menu while Server is offline.	IPv4 and Port number	IPv4 - String Port number - String of positive integer Example: IPv4 - 26.66.216.133 Port number - 5555	Failed to connect to server while error message show on terminal or popup show on screen for both Host/Join.	Error message on popup or terminal should be like: "Unable to connect server"	Passed		Passed		Passed		Passed		Passed	
3	Make Connection with invalid input	2	Attempt to make a connection with button Host/Join in Main Menu and put invalid type of inputs. [Sever isn't nessessary]	Any form of String	Example: IPv4 - 11.111.111.11 Port number - AAAN IPv4 - 12.123.12.12.1 Port number - Sand	Failed to connect to server while error message show on terminal or popup show on screen for both Host/Join.	Case IPv4 be any String and Port number be String of positive integer: Error message on popup or terminal should be like: "Unable to connect server" Case IPv4 be any String and Port number is NOT String of positive integer: Error message on popup or terminal should be like: "Port number must be positive integer"	Failed	program will force shutdown after host/join game	Failed	program will force shutdown after host/join game	Failed		Failed		Passed	
4	Join while Match full	2	Attempt to make a connection to available local server with button Join in Main Menu while Match is full.	IPv4 and Port number that corresponding to server	IPv4 - String Port number - String of positive integer Example: IPv4 - 26.66.216.133 Port number - 5555	Successfully connect then disconnect that client from server and show error message or popup on screen.	Error message on popup or terminal should be like: "Match is full"	Failed		Failed		Passed		Passed	Warning message should be more descriptive	Passed	
5	Join while Match isn't created	2	Attempt to make a connection to available local server with button Join in Main Menu while Match isn't created.	IPv4 and Port number that corresponding to server	IPv4 - String Port number - String of positive integer Example: IPv4 - 26.66.216.133 Port number - 5555	Successfully connect then disconnect that client from server and show error message or popup on screen.	Error message on popup or terminal should be like: "Match is not created"	Failed		Failed		Passed		Passed	Warning message should be more descriptive	Passed	
6	Enter Option	1	Click Option in Main Menu.	-	-	Successfully enter Option Menu without any error.	-	Passed		Passed		Passed		Passed		Passed	
7	Quit game	1	Click Quit in Main Menu.	-	-	Successfully Quit the game without any error.	-	None		Failed		Passed		Passed		Passed	
8	Adjust Sound	3	Change volume of background music and sound effects in Option Menu.	Sound volume	Volume - String of positive integer Example:	Sound change according to adjusted volume.	-	None		None		None		None		Passed	

You can check the completed table in this link:
https://mailkmuttacth-my.sharepoint.com/:x:/g/personal/wagee_nanta_mail_kmutt_ac_th/ERiWB_3xofhItcKfUP5LzLcBomm5qm-5wgnU9awC5PdWJA?e=H0UXqu

Self-Evaluation

Overall

About the results of our project

We can say that we're all satisfied with the final look of our game. But still, it was far from what we'd initially imagined. There are many things that we cannot achieve in this project but also there are many things that we can finish on time and perhaps do better than what we initially thought.

First, let's talk about what we've imagined. Since we were inspired by the original Avalon board game, initially we decided to make our game be a social game with multiple lobbies that can be created at a time and everyone can choose to join any lobby that they want. Inside the lobby, players can walk around, customize their look and talk with each other whether they're using a chat box or a voice chat. Players can also interact with each other via Emote, some pose, or some action. And, the host can also change any setting available in the game with as few limitations as possible. After the game is started, most of the options that can be done in the lobby are also available in the game like a chat box, voice chat, emote, pose, and ability to walk. Later in the game, at the phase that team members already get approved by every player and need to do the mission by selecting success or fail, we decided we want some mini-game at this point to make it more interesting for players. Moreover, we want to have a few cutscenes throughout the game like when team members have permission to do missions or when the game ends. We also think about when a player lost the connection from the server during the game, we want our game to handle this situation by making other players wait for the player who lost the connection to be able to reconnect again then the game will proceed as normal. There is much and more little detail and effort that we put into designing this game but what is already explained covers the major part that we want our game to be, to show how huge we dream our game to be.

So, what can we achieve at the end of our project? From what we have imagined, it is too much for us to bring it out considering the given time. So later, we decided to reduce our functionality to the level that it still covers everything but not too complicated as the initial thought. The function that we cannot get it done for example we are unable to make our game to have an ability to create multiple lobbies, players can only create only one lobby and we didn't have any interface to tell the player if there are multiple lobbies available from multiple servers and we are fail to make our player interact with each other, there is no emote or any pose for player. The voice chat function got cut off, the only thing left is the ability to walk and chatbox. For inside the match, we need to get rid of the cutscene function and focus more on core game play. And, we are unable to create a mini game as we hope from the start. But other than that, we can achieve it all. The part that we think we do better than we thought is probably the function inside the lobby, the arrangement of the scene system and the feel of each display menu. The reason is when we take more time to do the same part with less functionality it is easier for us to organize and we can refine it until we are satisfied for example in the lobby, the function that allow player to change their appearance, at first we think this function will be troublesome but when we took time with it, it turn out to be one of the solid function of our game. The hardest part of this project in terms of technical is to maintain the game's performance and due with data being sent between networks at the same time. But still we can find the way out and do as best as we can to finish this project.

About the development process

To find the development process that suits our group is very hard because everyone has different work time, experience, personality and personal matters to attend. So, we have to design a new rule and process in order to work together. It took us a few hours to write the new rule and adapt the process from scrum as already explained in this documentation (Development process description). Although we have very strict rules and clear process planning, many things still don't go as they should be. There are many coincidences that occur throughout our work period like unexpected tests, quizzes, projects from another subject or even urgent personal business. This made our work differ a lot from the original plan, some meetings were postponed from the regular meeting time and some weeks, we did not even have any meeting occur. Even though we are able to maintain regular meetings on Saturday and are able to get a lot of work done in time. (More info on Project timeline)

What we have learn

Through this project we learn a lot of things whether in terms of software development skills, programming skills, communication skills, artistic skills, or teamwork. At the end of the project, each person has a noticeable change in skill. The skill that improves depends on which part that person is responsible for. We also learn that the most important software engineering technique is to choose the right software development process. Even though we already stated that many things don't turn out according to plan, if it is not because of this plan, we are unable to make it through. To think more carefully, if we select a popular process without considering our team's needs, like choosing to use the real Scrum, it would make our working process worse than it should be and maybe at some point, we may not be able to control it. So, that is why we think "Select the right process" helps a lot. Other than this, we also think that communication skill and personal ability is also important. Many times miscommunication causes a lot of work to do and lack of ability (programming skills, development skills) make work a lot slower but fortunately, the lack of ability can be solved with more practice.

Each team member

Nithi Piyaphonnarinthon 62070503430

- For me, I take part in the testing. And record the bug I found so the other can fix it. I am satisfied with how the project went out as well as it is.

Romtam Tanpituckpong - 62070503444

- I am satisfied with how the project got this far, the only dissatisfaction is that due to the inexperience and confusion in the language we used, which is python. And how late I corrected my confusion made it hard to look back and fix it because there are many lines that I work with due to my confusion. And decide to move on since it does not affect the project that much, it is just about how to private variables according to the best practice. I build the function to create the component where there are many public variables that never be edited on the other function.
- And my aspect as a team member, I think the biggest problem of our group is that the workload difference that individuals can handle makes some members carry some parts more than one should after dividing the task. Which affects the schedule of the plan. But every member tries their best to pull the project out. We were glad that we got this project as it is.

Wagee Jr. Nanta Aree - 62070503445

- As for me, I am quite satisfied with the outcome although I think I could do it better. I was responsible for designing and coding for the game UI. At first, I had a bigger plan for UI but due to my inexperience with python, it slowed down my work. I think I should have spent more time and dedication on this project than I did. I also think I should have discussed with team members more because I think sometimes I created variables that are not really necessary. However, I think this is a great experience for me and the overall outcome is good, compared to the time limit.
- As for others, I am really happy and enjoy working with this team. Most of them are responsible for their tasks, especially Chatchapon, who I think is the most dedicated person in the team. Actually, Romtam and Kantawit also did hard work and tried their best. Although their skills in coding are not as good as Chatchapon, they try to finish the given tasks and help the team as much as they can.

Chatchapon Sukitporn-udom - 62070503455

- I would say this is one of my proud projects. I have learned lots of things from this work especially, network and multi-thread programming. I also learn to work with my team and organize my work along with my team. I have put my best effort into this and keep up with every process of work. Although a lot of work doesn't go well according to plan, many times there are some unpredictable matters like urgent work, my friend needs to attend to their matter or I need to go do some other work. But with a good enough development process, it makes me able to control the situation and drive everyone to deliver the work on time.
- For other people on the team, each person may have different work time and capacity. This often affects project plans and causes many problems which are not that bad since they are able to solve for example, some week, some team members cannot do their task due to personal business so it just requires some work on rescheduling the plan and try to finish other parts instead. But anyway, they try their

best to deliver their work which I really appreciated. I really hope they will keep up their good work to the next project in the future.

Kantawit Chatdamrongmongkol - 62070503457

- I would say that I have learned a lot from this project, especially how to work in the team. This is my first time working by following the plan and framework, we pragmatically rule to work together but also flexible due to the other concurrent assigned work. For the project, I had been looking forward to seeing the finished game everyday when we were working on the project. I am actually responsible for the game audio part; to find,adjust, and set all sound effects to the game. At first I was struggling with the coding part because I do not have experience coding with Python or Pygame before. I tried to keep motivated all the time during working on this project, but practically we need to have enough rest to always keep motivation up. I have the opportunity to learn many things in this project with the suggestion and support from my team.
- I want to say thank you to all members, we have passed through many things together. Thank you for the good suggestion and volitation toward me and this work.