Team-B D2: Design Deliverable

This document outlines the requirements for the ‘UU-Game’ product. The UU-Game is a two-player, 4x4 board game with sixteen unique game pieces each comprised of four binary characteristics. Players take turns placing a piece on the board that has been selected by the other player with the intent to win the game by having four adjacent pieces in a straight line each sharing at least one binary characteristic. That is, for example, there are four pieces in either a horizontal, vertical or diagonal row that each have a characteristic at the same position out of four positions being in the same binary state of either True or False.

The game engine (GE) component is Agile developed in the Python3 programming language by a team of six developers in two sprint iterations over the course of approximately three and a half weeks and with a total team effort of over two-hundred hours. The Agile development process has involved a significant level of dedicated planning and design, as well as very thorough testing with over eighty successful unit tests. The design is comprised of an object pool for generating game entities, such as the game board and the piece pool, and an abstract factory pattern for instantiating either user-controlled or a variety of AI player classes. The GE computes game play functionality by either processing user-controlled game play commands or automating AI game play, and updates the game state at each game play turn until either the game is won by a player or the game ends in a draw.

There are three different AI game play difficulties – easy, medium and hard – which, whether playing as user vs AI or AI vs AI, each have a unique algorithm designed specifically for that difficulty and its difference from the other difficulties, providing a variation of user experience. An example of such, is a variant of a well-known algorithm used in game theory and artificial intelligence called Minimax. Implementing the Minimax, the hard AI algorithm, in theory, should not even be able to beat itself, let alone should be able to be beaten by a good player at the most of times. Whereas, in theory, the custom algorithm being implemented for the easy AI difficulty should be winnable against by even the most inexperienced of players, while the medium AI difficulty should be beatable at approximately fifty-percent of the time for players at any level of ability.

The GE system is Agile developed in the Python3 programming language with dedicated team planning, good class design practices, efficient implementations, such as list comprehensions, descriptive documentation and an abundance of thorough testing.

1. Glossary:

Piece: A token with a unique set of four binary characteristics. There is a total of 16 pieces.

Board: A four by four field where pieces can be placed on. There can only be one piece in each slot of the field.

Turn: A turn consists of player one choosing an existing piece, player two then places the piece on a free slot on the board.

Slot: Space on the board which unique coordinates, that can be occupied by up to one piece.

Binary: Data type with only two possible states.

Value: State of the two possible ones of binary type.

Characteristic: Attribute for a piece with binary value.

Component: Subsystem of the overall game. For the UU-Game there are three different ones. The Game Engine, Game Platform Communication Platform.

Player: Ether human or computer control entity playing the game.

User: Player controlled by the input on the keyboard.

2. Definitions:

Choose Piece:

Select one of the remaining game pieces with a unique set of characteristics. This is the piece the opponent must play in this turn.

Place Piece:

A Player must place the piece given to him by the opponent during this turn on one of the remaining free slots on the board.

Win Condition:

A Player wins the game when he places a piece that leads to a board state where four pieces in a row, column or diagonal share the same value for at least one of the four binary characteristics.

AI:

The game engine AI can play as an opponent to either the human player or another AI. These AIs come in three different difficulties; easy, medium and hard. The easy difficulty is intended for new players and can be beaten by understanding the basic game rules. The medium difficulty is intended for players who know all the game rules and should provide a challenge for intermediate players. The hard difficulty is playing extremely well and is difficult to beat by human players but doable by masters of the game.

3. Components:

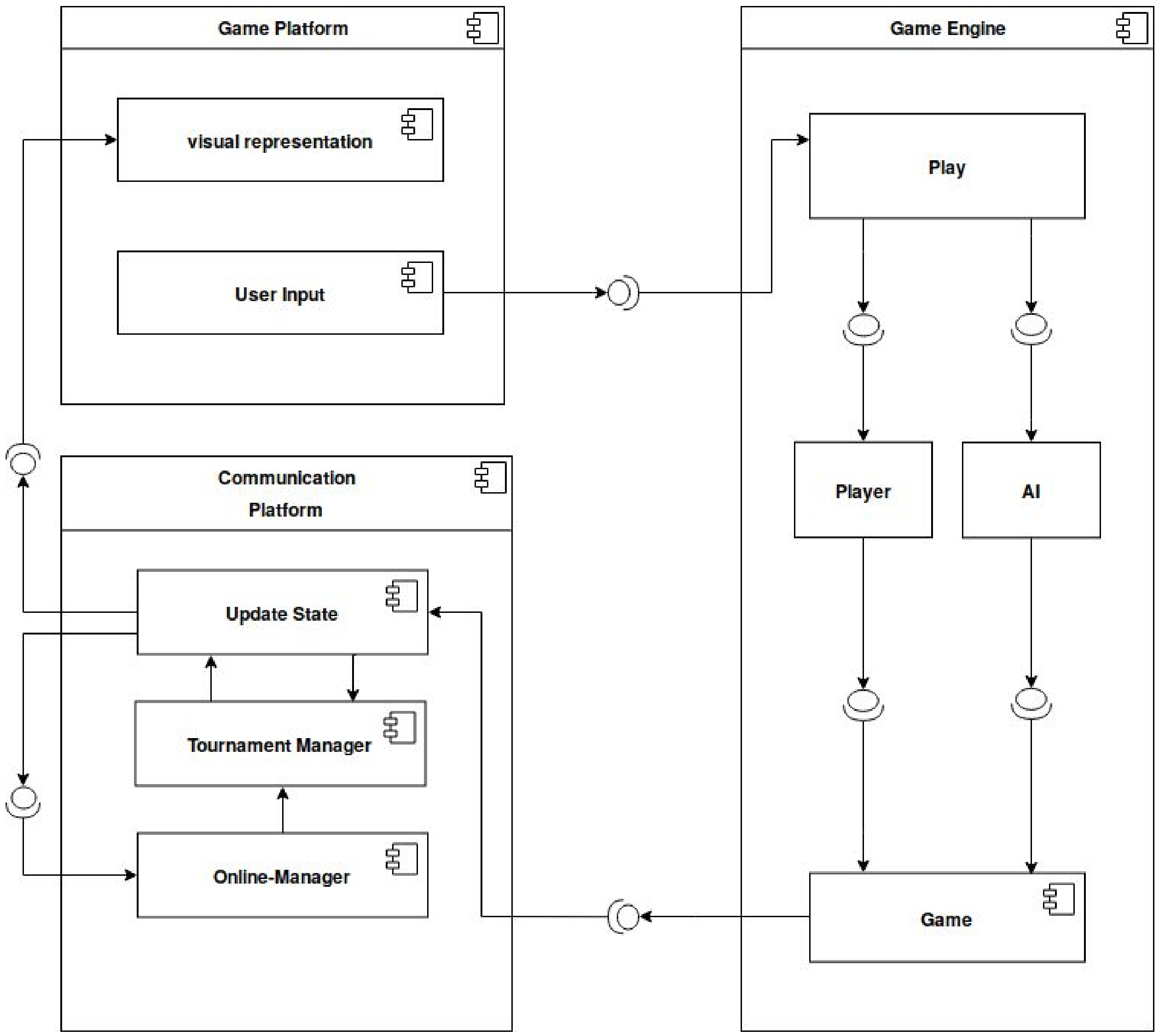


Figure 1: Component diagram for the UU-Game.

3.1 Game engine (GE)

Computes the logic and functionality of the interactions between each player and the UU-Game. Is comprised of an entity class for the game logic, and player abstract concrete classes for the game functionality. Generates the game board and each of the sixteen unique pieces at initialisation and updates the game status at each play of the game. Either processes input from a user player for each play of the game, or automates game play for each of three AI classes of differing easy, medium and hard difficulties.

3.2 Game Platform (GP)

Acts as a platform between the communication platform and the game engine components. Initialises players and the game, computes local and online game play turn logic, translates player input from the CP to the GE, and generates a visual physical representation of the UU-GAME status including board and pieces at each play of the game. Is comprised of a platform class for play logic, game and player initialisation, and inter-component communications, a diagram class for the game imagery, and a piece entity class for generating piece imagery.

3.3 Communication platform (CP)

Prompts and processes user commands for game play selection. Enables user-system and host-client communications for local and online gaming. For each of local and online gaming, provides 1 vs 1 and tournament game functionality for any variation of user and AI players between two players for 1 vs 1 and up to eight players for tournament. Comprised of a platform class for providing menu options, and setting up each type of game and game settings.

3.4 Sub-Components

Game:

This is where the state of the board and the remaining pieces of the current game are kept.

Play:

This component handles the logic of letting the different players (ai or user) take turns choosing and placing pieces.

Player:

There can be between zero and two human players in a game. They get a dialog to interact with the game engine to perform the choose_piece and place_piece functions. If an invalid input is given they are asked again.

PlayerEasyAI:
Choose a piece by checking every empty position for a worst position. Always gives away a winning piece if available. Places a piece by checking for the worst placement. Never takes a win unless forced.

PlayerMediumAI:
Choose a piece by checking every empty position for a best position (a win) and then implements the best position 50% of the time. Places a piece with the same principle, 50% chance of taking a win that is presented to it.

PlayerHardAI:
The implementation of the hard AI is using a well-known algorithm used in game theory, named Minimax. The algorithm builds an evaluation tree of a specified depth, trying all possible placements of a piece as well as passing a piece that will not result in the opponent winning. Making the game challenging even for an experienced user.

4. Functional requirements:

User Story:

1. As a user I want to start a graphical 4x4 board game that has sixteen unique game pieces with four different binary characteristics so that I can see the status of each game at each stage of the game play.

2. As a user, I want to be able to start a game as either user vs user, user vs AI, or AI vs AI so that I can play a game with either type of game play setting. All this options are presented when starting a new local game.

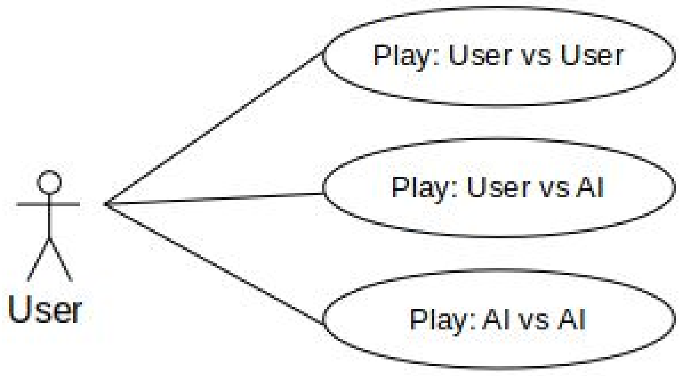


Figure 2: Game play choices for a player in local mode.

3. As a player I want to have a game play choice of an easy difficulty for AI that should be easy to win against by even the most inexperienced of players when playing the game as human player vs AI player. As seen in Figure 3 this Option should be available for Human vs AI as well as AI vs AI.

4. As a player I want to have a game play choice of a medium difficulty for AI that wins on average approximately 50% of the time when playing the game as human player vs AI

player. As seen in Figure 3 this Option should be available for Human vs AI as well as AI vs AI.

5. As a player I want to have a game play choice of a hard difficulty for AI that wins most of the time when playing the game as human player vs AI player. As seen in Figure 3 this Option should be available for Human vs AI as well as AI vs AI.

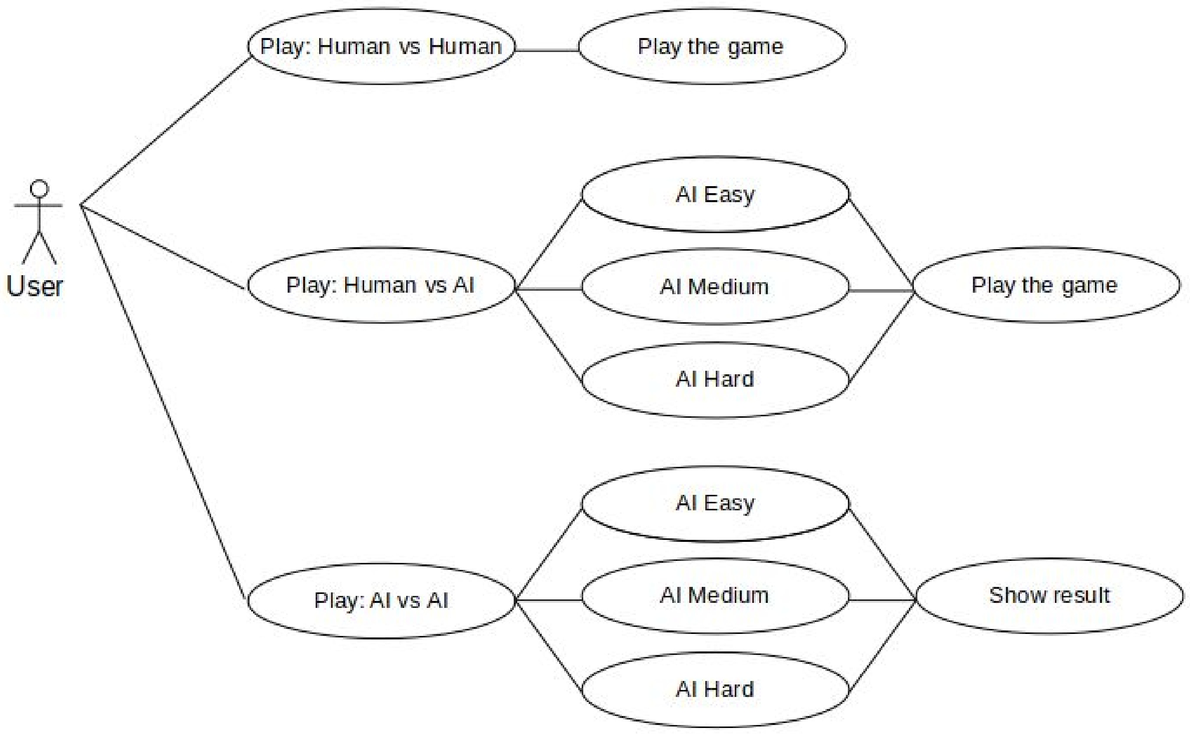


Figure 3: Selection tree for local game with AI and difficulty selection.

6. As a player, I want the starting player-turn to be randomly selected so that I don’t have to decide who starts a game. This is valid for all playable gamemode defined in Figure 3.

7. As a player, I want to be able to select a game piece that is not already on the board so that I can offer it to the other player for placing somewhere on the board during their play-turn.

8. As a player, I want to be able to place a pre-selected game piece somewhere on the board that does not already have a piece allocated to it so that I can try to win the UU-Game.

Figure 4 presents the basic flow of a round. At the beginning a check is made to determine if there is still space on the board and therefore the game has not concluded. If there is space on the board depending on which turn it is either Player_1 or Player_2 starts by choosing a valid piece followed by the opponent placing this piece at which point the Board is updated and the cycle starts again.

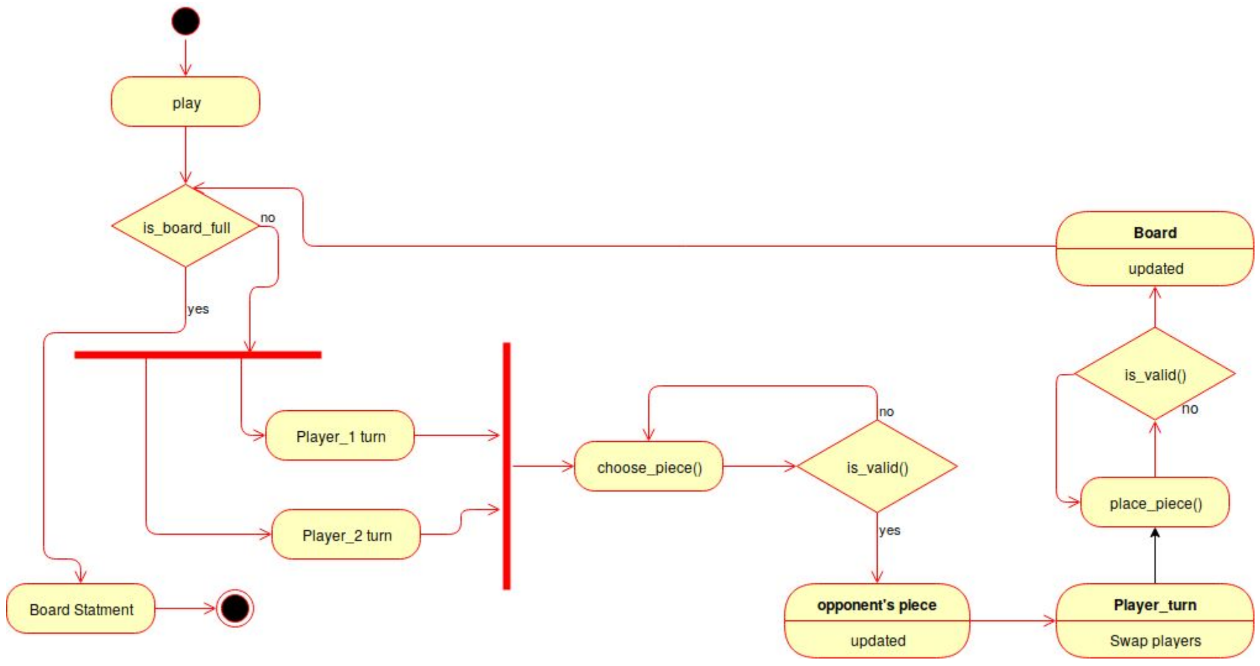
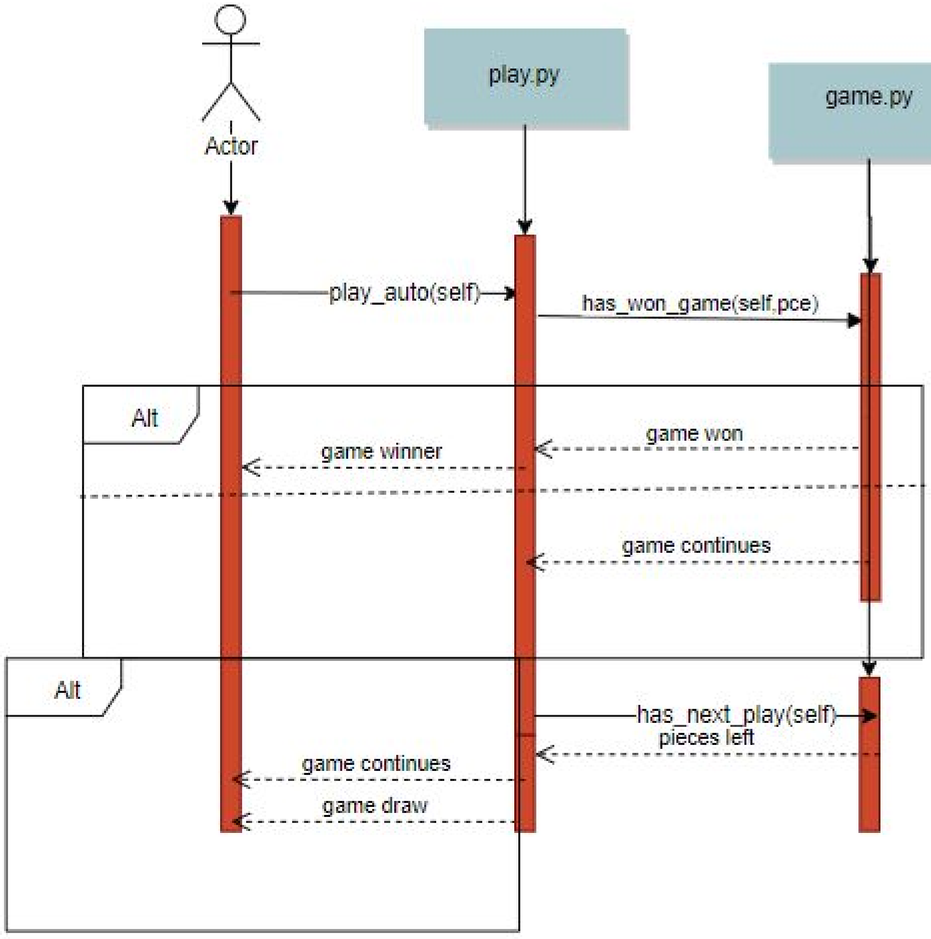


Figure 4: Basic activity diagram of a turn in the game.

9. As a player, I want the game to stop when there are four game pieces in a row in either a horizontal, vertical or diagonal direction each sharing at least one of the four binary characteristics so that I know if and when a game is won.
10. As a player, I want the game to stop when there are no more available spaces left on the board for placing a game piece and the final piece placed on the board did not result in a win so that I know when the game has ended in a draw



.

Figure 5: Check for win condition.

11. As a user I want to only view the end game result when I choose the option of an AI player vs AI player game. In contrast to the normal game flow the intermediate steps of

the game are not shown. As shown in Figure 3 this applies to all AI vs AI games regardless of the selected difficulty.

12. As a user, I want to have a selection of game play options so that I can choose the type of game that I want to play. This includes the following options. First local or online gameplay. Second single game or tournament mode. As seen in Figure 6 these options are available either in local or online mode.

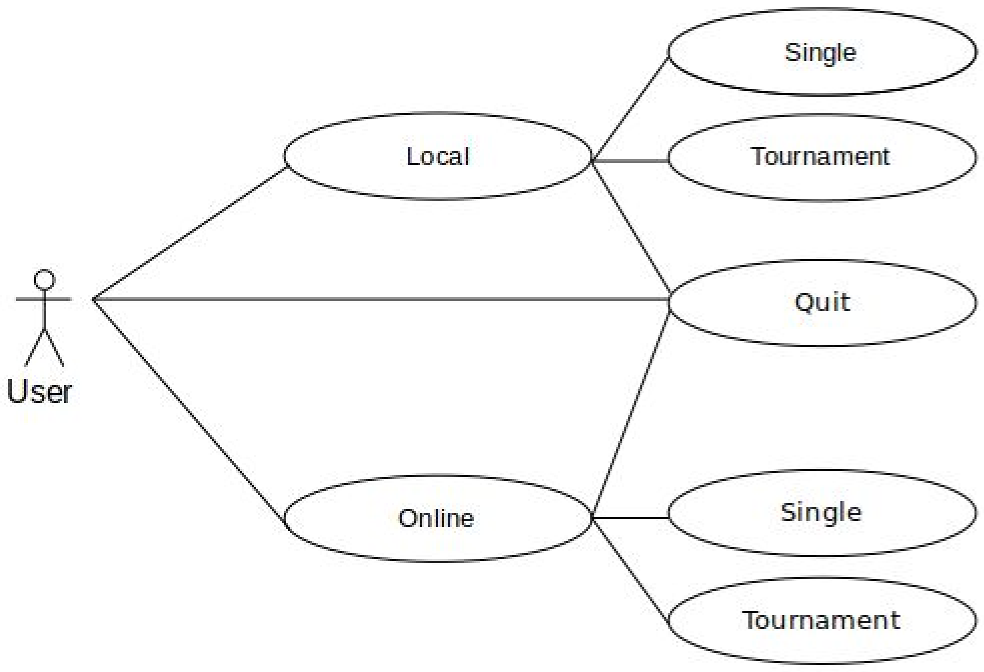


Figure 6: Game selection menu.

13. As a user, I want to be able to start a local tournament game of any combination of AI or user-controlled players so that I can play tournament games against other players

locally. The games will all be played on the local machine and the current players are displayed.

14. As a user, I want to either host or join an online tournament game of any combination of AI or user-controlled players so that I can play tournament games against other players online. The game will be played on two computers connected by a local network, the different matches of the tournament will be played after one another and the current match and players will be displayed at the beginning of each game.

15. As a user, I want to be able to start an online ‘singles’ (1 vs 1) game of any combination of AI or user-controlled players so that I can play a ‘singles’ (1 vs 1) game against another player online.

16. As a user, I want to have an option available to me while implementing the UU-Game menu and not playing a game so that I can quit the application at will. As seen in Figure 6 this options is available during the whole time in the game selection menu.

17. As a user, I want to be able to view the type of game, the type of players (user or AI), their provided names and, if AI, the difficulty so that all players can know the details of the game being played.

18. As a user, I want to be able to view a graphical representation all available pieces so that I can distinguish and select a piece for the other player to place on the board during their play-turn.

19. As a user, I want to be prompted to enter specified commands for selecting a piece so that I can select a piece for the other player to place on the board.

20. As a user, I want to be able to distinguish a graphical representation of the current selected piece for placing on the board so that I know which piece that I am to place on the board when it is my turn to play.

21. As a user, I want to be able to distinguish an available place on a graphical representation of the current board status so that I can choose where to place a pre-selected piece.

5. System constraints:

The system that the program is being run on requires being able to support Python 3.7. Python is a cross-platform programming language, enabling the product to be run on either a Linux, Mac or Windows PC. However, Python may be required to be installed, or at least updated to version 3.7 prior to running the program on any PC. The minimum PC system requirements are estimated at being single-core 1 Ghz CPU, 64 MB graphics card, 1 GB RAM and 40 GB hard drive. The program itself requires less than 1 MB of storage space. The program also requires a terminal for the CP.

6. Non-functional requirements:

Development requirements: With the written consent of the owners of this project, further development is allowed.

Regulatory requirements: This developed software must not be distributed unless with written consent of the owners.

Space requirements: The program itself requires less than 1 MB of storage space. Additional space for Python 3.7 and supporting software is required.

Usability requirements: The game responds at least after five seconds. This limit is set specifically for the computation of the AI algorithm.

Security requirements: The game does not keep a history of previous games and does not save them internally. The only exception is for the duration of a tournament the outcomes of the games of the tournament are stored in the bracket overview.

Reliability requirements: The online gameplay is only available on local networks. Both computers have to be connected to the same local network in order to play the game together.

Robustness requirements: The game provides the option to reconnect to an online game after a disconnect happened for a duration of 5 seconds.

Performance requirements: Python 3.7 or above has to be installed on the system. The program requires a terminal for the CP and a keyboard as input device.

AI constraints:

- In theory, based on user requirements, Easy AI should be beatable at the most of times by even the most inexperienced of users. Due to this requirement being difficult to test with a lack of experimental users and time, it was decided to design the EasyAI to have an average win rate of no more than 10% against the other AI difficulties.
- In theory, based on user requirements, Medium AI should be beatable at approximately 50% of times by users of all abilities. Due to this requirement being difficult to test with a lack of experimental users and time, it was decided to design the MediumAI to have an average win rate of between 40% and 60% against the other AI difficulties.
- In theory, based on user requirements, Hard AI should not be beatable at the most of times, if not at all times by users of any ability. Due to this requirement being difficult to test with a lack of experimental users and time, it was decided to design the HardAI to have an average win rate of no less than 90% against the other AI difficulties.