Backgammon

JJ Style

February 23, 2018

Contents

1 Analysis

1.1 Problem Definition

For my project I will be creating backgammon. I will be creating this for my client, Mr Ovayolu, who wants to encourage his younger students to learn to play more classic board games. My game will involve an easy and hard AI for the children to practise against, as well as a 2 player mode to play against each other. It will also include a leaderboard to motivate the children to play, practise and compete more.

1.2 Research

1.2.1 Interview

The interview with my client will be very important so I can find out what they want from the finished version of the game. I can analyse the clients response so I can tailor the game to their requirements.

1.2.2 Questionnaire

I will develop a questionnaire using google forms which can allow me to gather a lot of quantitative data quickly from my end users. I will send my online questionnaire to Mr Ovayolus year 7 students. The questionnaire will include questions about different features that they would or would not like to see in a game of backgammon.

1.2.3 Current Versions

In my research, I will also evaluate old and current digital backgammon games, both online and on mobile platforms. This will allow me to see what the standard features are in every game and also additional features in the game which the client can decide on during an interview.

1.2.4 Artificial Intelligence in Video Games

Artificial intelligence may take form in 2 ways. True AI which facilitates computer learning, and automated computation where the computer follows a predetermined set of rules based on a set of inputs. One of the first ever

examples of AI in a game was made in 1951 for the game Nim. In the same year, AI was introduced to both checkers and chess. These developed over time becoming better and more advanced and in 1997, IBMs Deep Blue AI beat chess grandmaster Garry Kasparov. AI is becoming so advanced, soon it may be possible that it will be more intelligent than the human brain.

1.3 Background to Project

Backgammon is one of the oldest board games originating from Persia around 5000 years ago. It is traditionally played by 2 people who take it in turns to try and remove all their pieces from the board first. The movement of pieces is determined by the rolling of 2 dice. It is a game that requires both luck and tactics.

1.3.1 First Internet Backgammon Server (FIBS)

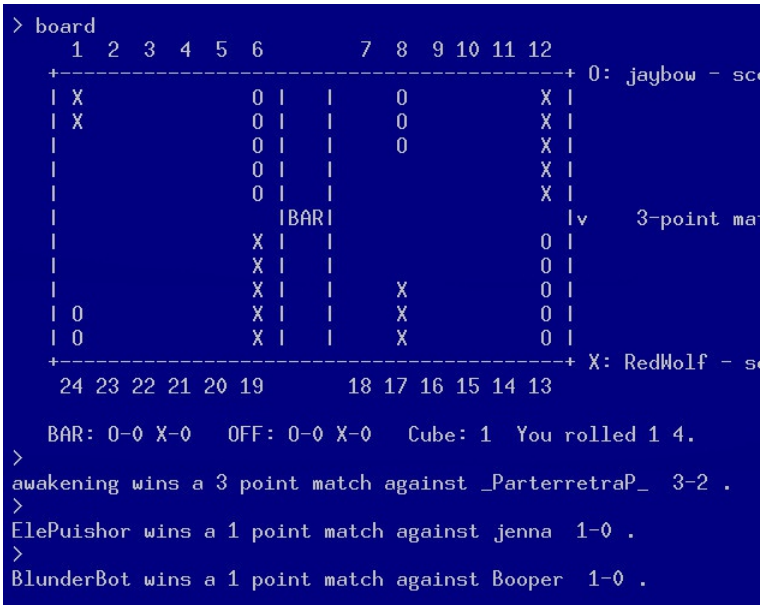


Figure 1.1: Screenshot from FIBS, originally developed in 1992, it was the first online backgammon game you could play in real time. The game was played in a command line interface and the board and pieces were displayed in ascii characters.

1.3.2 Backgammon application on Android and iOS by AI Factory Limited

I like this backgammon application because it allows users to customise the set-up of the game such as the difficulty of the AI or which colour piece you are. It even includes the option to play with the Crawford Rule which I have not seen commonly among other backgammon games. Furthermore, it allows even more customisation in the settings; for example choosing which way the pieces move round the board - clockwise or anticlockwise and whether or not to highlight valid moves.

Figure 1.2: Backgammon mobile application

Pros

1. Aesthetically pleasing - simple but not too bare
2. Highlighted legal moves is very useful
3. Undo button
4. Scores are well grouped together
5. Settings changeable from within game
6. Lots of customizable settings

Cons

1. Hypothesised cheating AI seems to roll an extraordinary amount of high doubles just when it seems to need them most

1.3.3 Backgammon Online

Figure 1.3: Source: `http://games.aarp.org/games/backgammon/`

Pros

1. Undo button
2. Quick option to enable and disable sound on the side panel
3. Access to brief instructions on the side panel
4. Highlighted legal moves is very useful
5. Spinning highlighted pieces signifying whose turn makes it very clear

Cons

1. No customising at all in the menu or in the settings
2. Pieces feel too big
3. Pip score for each player is separated on each side making it harder to quickly analyse who is winning

1.4 Interview

1.4.1 Questions

1. What age group is this game of backgammon for?
2. What standard features do you expect to see in a game of backgammon?
3. What additional features would you like to see (undo button, leaderboard, highlighting legal moves, instructions etc.)?
4. Would you like the additional features to be optional in the settings?

1.4.2 Answers

1. 11 to 12 years old (Years 7 and 8)
2. Most importantly just a simple, fully functional game that you can play either 2 player or against an AI. I think it is important to be able to save the game so you can resume it later because not everyone has time for a full game.
3. An undo button would be nice as it allows the students to correct their mistakes. Also highlighting the valid moves is useful to help them learn where they are allowed to go. Also an instructions screen would definitely be helpful for students learning the game.
4. I think these features should all be optional so as the students develop they can remove the assisting features.

1.4.3 Analysis

From this interview I have learnt that I am making the game for quite a young audience so it is important that I include helpful features like an undo button and highlighting legal moves however these can be switched off in the settings if required. Furthermore as the clients are young and just learning the game, I should be careful not to make my AI too difficult otherwise they will lose more and feel less motivated to practise and play. However, the AI will become too easy as they progress therefore I will include different difficulty levels for my AI, easy and hard.

1.5 Questionnaire

1.5.1 Responses

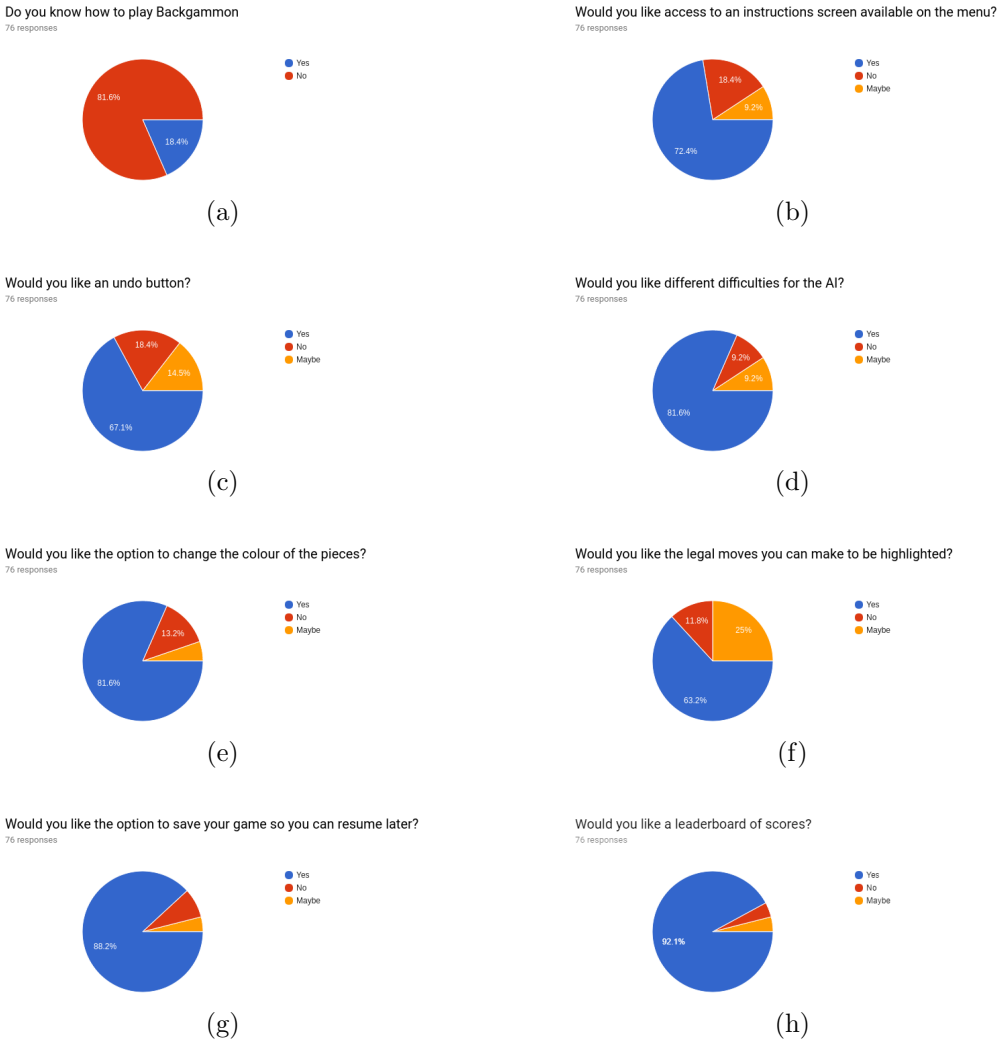


Figure 1.4: Backgammon questionnaire responses

1.5.2 Analysis

My questionnaire was answered by a large sample - 76 students from years 7 and 8. From my questionnaire I found that many of my end-users do not know how to play backgammon already. This was a critical piece of information to learn about as it informs me that I must tailor the game for people just starting to learn the game. It was therefore unsurprising to find that the majority of people would find it useful to have access to an instructions/rules page. So this is something that I think is necessary to include.
Regarding game features, all were viewed as desirable, however being able to save your game and having a leaderboard had a noticeably larger majority in favour of them. Therefore, I think it is important to ensure these features are added to the game. Other features, such as highlighting valid moves, had a moderately balanced split between yes and no. To cater for everyone's preference I will add all the features but have the option to switch them off in the settings if not desired.

1.6 Objectives

1. Create a menu
    (a) Buttons to start new game, load game or quit
    (b) Buttons highlight when hovered over
    (c) Buttons for instructions, leaderboards and settings screen
    (d) Instructions, leaderboards and settings have access back to main menu
2. Create the game
    (a) Pieces and destinations are selected using the mouse
    (b) Players should be able to undo their moves
        i. Player can undo moves
        ii. Can only undo the 5 most recent moves
    (c) Settings and instructions should be accessible during the game
        i. Dice animation, piece colour, ai pause, highlight valid moves should be changeable from within the game

ii. AI difficulty and 1/2 player should *not* be changeable in the settings once the game has begun

3. Rules for player and AI should be the same
   (a) Must move a piece off the bar first if you can
   (b) Can only select a piece of your colour
   (c) Can only select a legal destination - Destination is legal if:
      i. Destination has no pieces OR
      ii. Destination has any number of your pieces on OR
      iii. Destination has only 1 piece of the opposing colour OR
      iv. Destination is the bear off bar if all pieces are in the home quarter
   (d) If a double is rolled, player has 4 moves of the number that the double is
4. Create an AI with different difficulty levels
   (a) Easy AI will make random moves
   (b) Hard AI will make more safe moves and have different game strategies.
5. Create a leaderboard scoring the winner with the number of moves made to win the game
   (a) Users should be able to input their name after the game
      i. name must consist of 2 words. Each with length ¿= 1
      ii. name can only contain one space
   (b) Scores will be saved in a csv file which will be appended to
   (c) User can choose to sort the leaderboard
      i. most recent scores
      ii. first name
      iii. last name
      iv. score
   (d) User can order leaderboard in
      i. ascending order
      ii. descending order

1.7 Feasibility

1.7.1 Programming Language

I am going to programme the game in python as it is a language I am already familiar with. I will display the graphics of the game using Pythons pygame module.

1.7.2 Acceptable Limitations

I have spoken to my project supervisor and have decided not to include music and sound effects because it could be distracting to some students when initially learning to play and thinking about what moves are best to make. Also we didnt think piece animation is necessary as this project is focused on learning to play and not game experience.

1.7.3 Hardware and Software Requirements

Table 1.1: Table of requirements for my project

Hardware/Software	Justification
Python 3.X with the pygame module installed	My project will be coded in python so python along with the pygame module will be needed in order to run the game.
Keyboard and mouse	Will be needed for user inputs in the game
Computer monitor with a minimum resolution of 700 x 500 pixels	This is the size of the window of my game. If the resolution is smaller the game will not fit on the display

1.8 Modelling

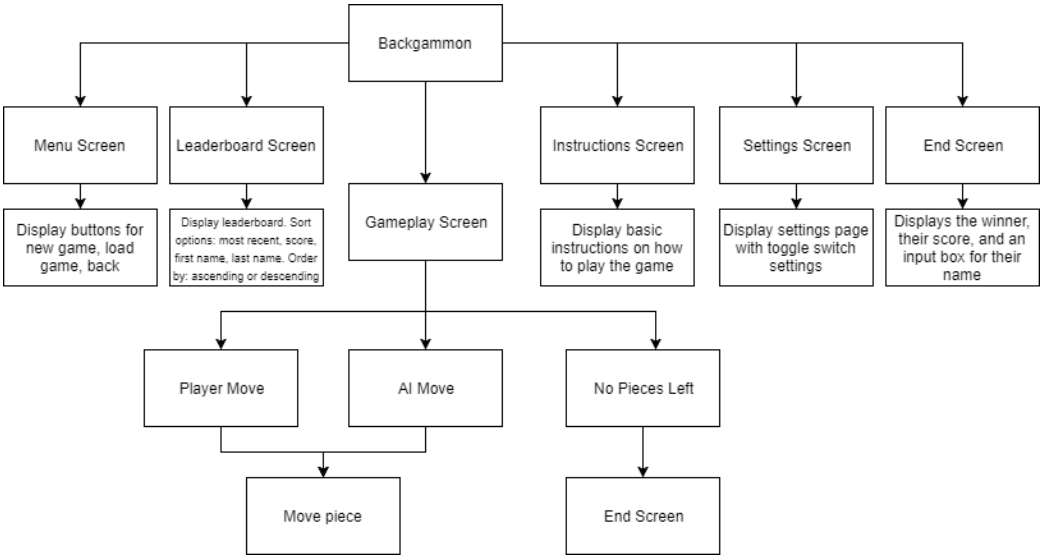


Figure 1.5: General modelling of the project.

1.8.1 Key Variables

Table 1.2: Table of some key variables which will be included in my game.

Variable	Data Type	Validation	What it controls
Board	2 dimensional array	None	Main backgammon board. Contains spike classes.
initialise_type	String	None	Specifies whether to load saved game or play new game.
SAVEDGAME	String	None	Stores csv filename to load and save saved game from.
winner	String	None	Stores the winner of the game.
score	Integer	None	Stores the winner's score.
winner_name	String	See page 21	Players name to be added to leaderboard.
GameOver	Boolean	None	Stores whether the game has finished.

2 Design

2.1 Design Mockups

(a) Buttons to play a new game, load a saved game or quit

(b) Settings button directs user to settings screen where game functions can be controlled

(c) Instructions Screen displays a translucent rectangle over screen displaying basic instructions

(d) Leaderboard Button directs user to leaderboards screen showing rankings

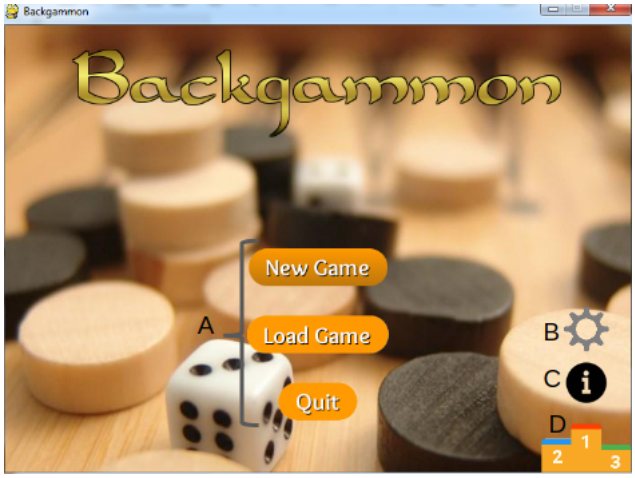


Figure 2.1: Main menu screen

Table 2.1: Table of objects on figure 2.1.

Object	Type	Settings
New Game	Button	Colour = (255,153,0) Text = "New Game" Font = Calibri
Load Game	Button	Colour = (255,153,0) Text = "Load Game" Font = Calibri
Quit	Button	Colour = (255,153,0) Text = "Quit" Font = Calibri
Settings	Button	Image type = .png
Instructions	Button	Image type = .png
Leaderboards	Button	Image type = .png
Background	Image	Image type = .png

(a) Toggle switches to switch off / change certain features.

(b) Back button to return to menu or game.

(c) Certain settings e.g. AI difficulty cant be changed from within the game, only the menu. If attempted to change players will be met with a pop up error message as seen below in figure 2.3.

Figure 2.2: Settings screen

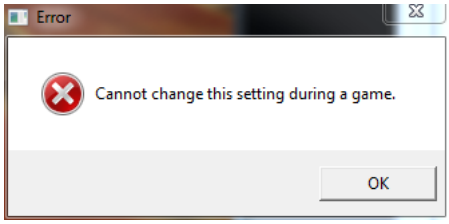


Figure 2.3: Error message in settings

Table 2.2: Table of objects on figure 2.2.

Object	Type	Settings
Highlight Moves	Toggle switch button	On = display on image + text = “highlight moves on” Off = display off image + text = “highlight moves off”
Black vs White	Toggle switch button	On = display on image + text = “Black vs White” Off = display off image + text = “Blue vs Red”
Hard AI	Toggle switch button	On = display on image + text = “Hard AI” Off = display off image + text = “Easy AI”
Dice Animation	Toggle switch button	On = display on image + text = “Dice Animation On” Off = display off image + text = “Dice Animation Off”
AI Pause	Toggle switch button	On = display on image + text = “AI Pause On” Off = display off image + text = “AI Pause Off”
1 Player	Toggle switch button	On = display on image + text = “1 Player” Off = display off image + text = “2 Player”
Back button	String	Colour = (255,255,255) Text = “Back” Font = Calibri
Background	Image	Image type = .png

(a) Panel on the left allowing custom sorting of leaderboard

(b) Option to sort by ascending and descending

(c) Highlight the selected options in green so easy to see what it is showing

(d) A back button to return back to the main menu

(e) A translucent backdrop behind leaderboard so it is easier to read

Figure 2.4: Leaderboard screen

Table 2.3: Table of objects on figure 2.4.

Object	Type	Settings
Sort by's (firstname,surname,score,most recent)	String	Colour = 255,255,255 Selected then Colour = (0,205,0) Font = Calibri
Order by's (ascending, descending)	String	Colour = 255,255,255 Selected then Colour = (0,205,0) Font = Calibri
Names	String	Text = firstname last-name score Colour = 255,255,255 Font = Calibri
Backdrop	Pygame Surface	Colour = (211,211,211) Opacity = 75
Back button	String	Colour = (255,255,255) Text = "Back" Font = Calibri
Background	Image	Image type = .png

(a) The board consists of spikes which can be selected

(b) The dice rolls are shown

(c) Players pip score and turn is highlighted at the top of the board

(d) Settings, undo and instructions are accessible on the side

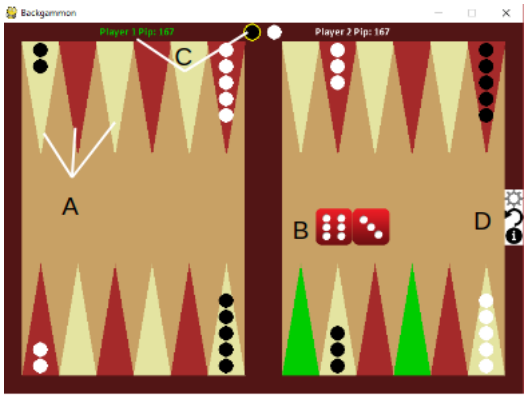


Figure 2.5: Game screen

Table 2.4: Table of objects on figure 2.5.

Object	Type	Settings
Spike	Pygame Polygon	Colour = (165,42,42) (dark), (228,228,161) (light), (0,205,0) (green)
Dice	Image	Image type = .png
Pieces	Pygame Circle	Colour = (0,0,0) (255,255,255)
Pip Score	String	Text = “Player P pip score xyz” Colour = (255,255,255) (0,205,0) Font = Calibri
Settings Button	Button	Image type = .png
Undo Button	Button	Image type = .png
Instructions Button	Button	Image type = .png

(a) Name entry field

(b) Winners score shown

Figure 2.6: Game over screen

Table 2.5: Table of objects on figure 2.6.

Object	Type	Settings
Background	Image	Image type = .png
Winner and Score	String	Text = "<textWinner>Wins! Score <score>"
Winner name	String	Text = "firstname lastname"

Validation for name input

Listing 2.1: The set of valid names expressed in Backus Naur Form

```
<letter> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<word> ::= <letter>|<letter><word>
<space> ::= ' '
<name> ::= <word><space><word>
```

The names that are allowed to be entered into my system can be described as any two words consisting of one or more characters with a space in between.

(a) instructions during game

(b) Pre-rolled state in game

(c) Rolling a double and bearing off

Figure 2.7: Other game screens

2.2 Technical Diagrams

2.2.1 Sorting Algorithms

Listing 2.2: Merge sort pseudocode algorithm

```
1  SUB merge (array_a, array_b, order_by)
2  merged <- []
3  WHILE length(array_a) <> 0 AND length(array_b) <> 0 DO
4      IF array_a[0][order_by] < array_b[0][order_by] OR
          array_a[0][order_by] = array_b[0][order_by] THEN
5          merged.append(array_a.pop(0))
6      ELSE IF array_a[0][order_by] > array_b[0][order_by]
           THEN
7          merged.append(array_b.pop(0))
8      ENDIF
9  ENDWHILE
10 IF length(array_a) <> 0 and length(array_b) = 0 THEN
11     merged <-  merged + array_a
12 ELIF length(array_a) = 0 and length(array_b) <> 0 THEN
13     merged <-  merged + array_b
14 ENDIF
15 RETURN merged
16 ENDSUB
17
18 SUB mergesort(array, order_by)
19     IF length(array) = 0 OR length(array) = 1 THEN
20         RETURN array
21     ELSE
22         int middle <- length(array) / 2
23         array_a <- mergesort(array[:middle], order_by)
24         array_b <-  mergesort(array[middle:], order_by)
25         RETURN merge(array_a, array_b, order_by)
26     ENDIF
27 ENDSUB
```

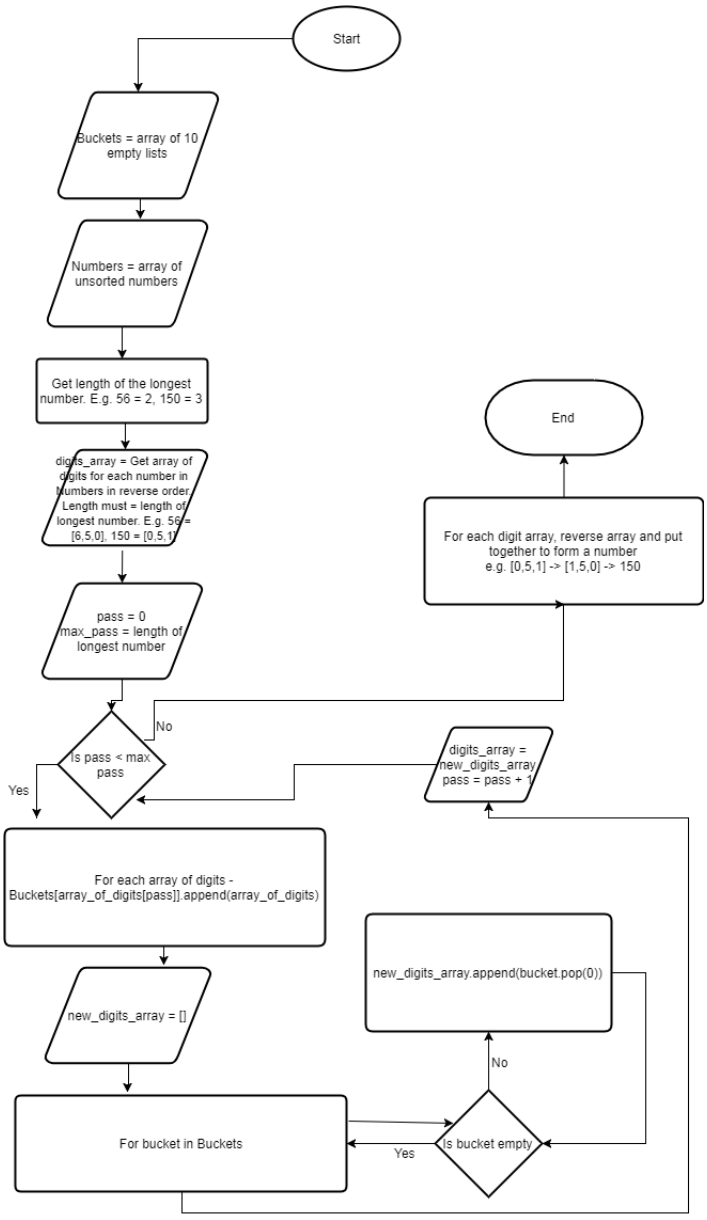


Figure 2.8: Radix sort flow diagram algorithm

Explanation for Different Sorting Algorithms
After doing some research on the time complexity of different sorting algorithms, I decided to use 2 different sorting algorithms in sorting my leaderboard. The first is a radix sort, which I use to sort the leaderboard by score (integer sorting). This is a non-comparative sort which is very efficient, with a big O of $O(nk)$, where n is the number of integers to sort and k is the length of the longest number.
I then decided to use a merge sort for lexicographically sorting the leaderboard - alphabetical order. This is because the radix sort is a comparative sort and therefore it is simpler to program as a mergesort as I can just compare each string with less than or greater than. The mergesort was chosen as it also has a fairly good time complexity of $O(nlogn)$.

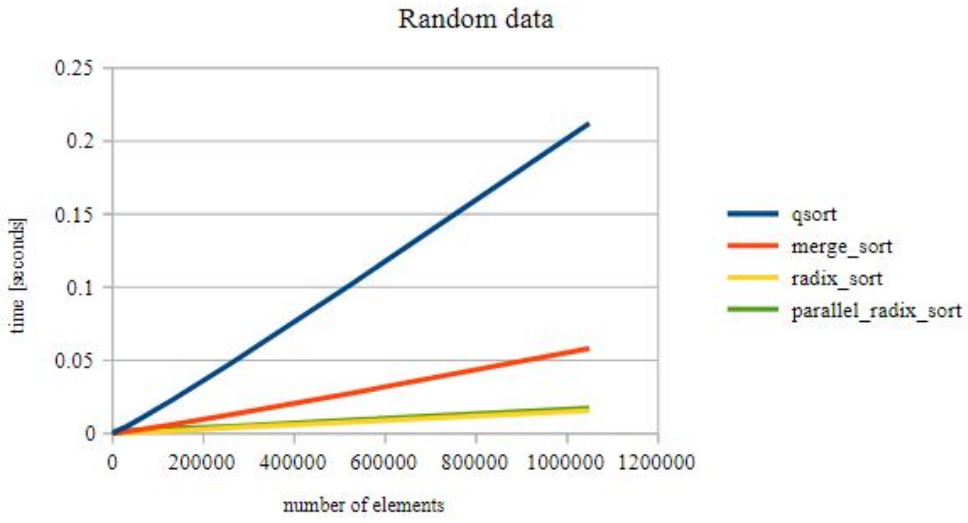


Figure 2.9: Graph of time taken to sort random data using different sorting algorithms

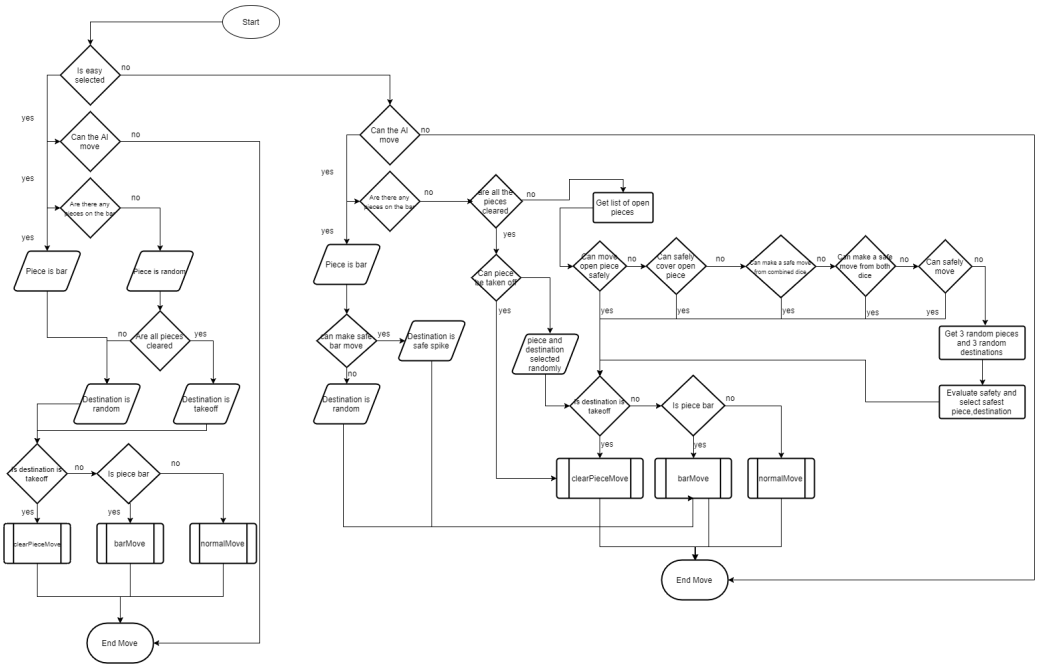


Figure 2.10: AI algorithm flow diagram

The easy AI simply makes a random move every turn. The hard AI passes through a number of checks, to see if there are potential safe moves available. Safe moves found are appended to a list and a random safe move is chosen. This aims to reduce the chance of a piece being taken.

2.2.2 Class Diagram

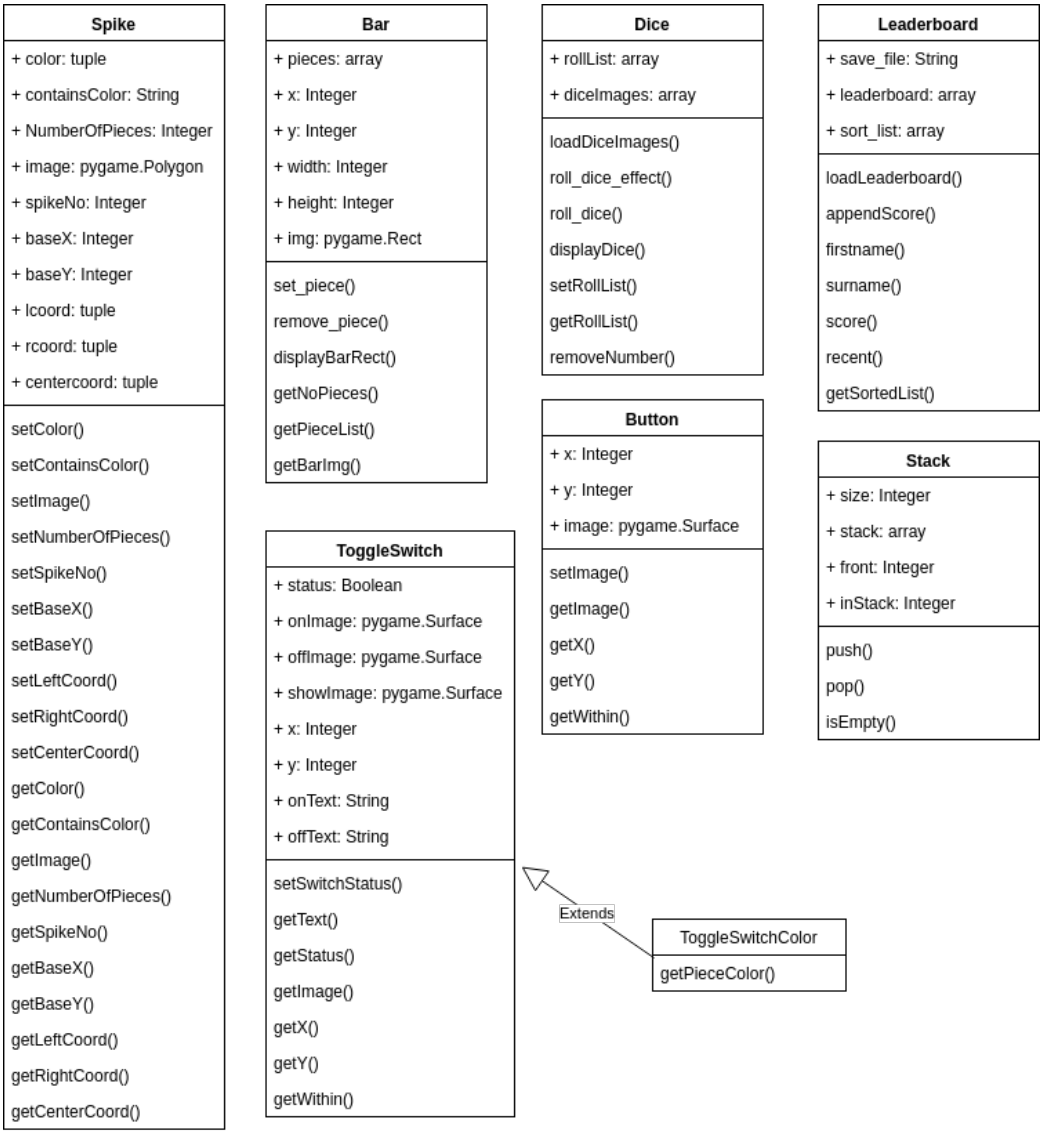
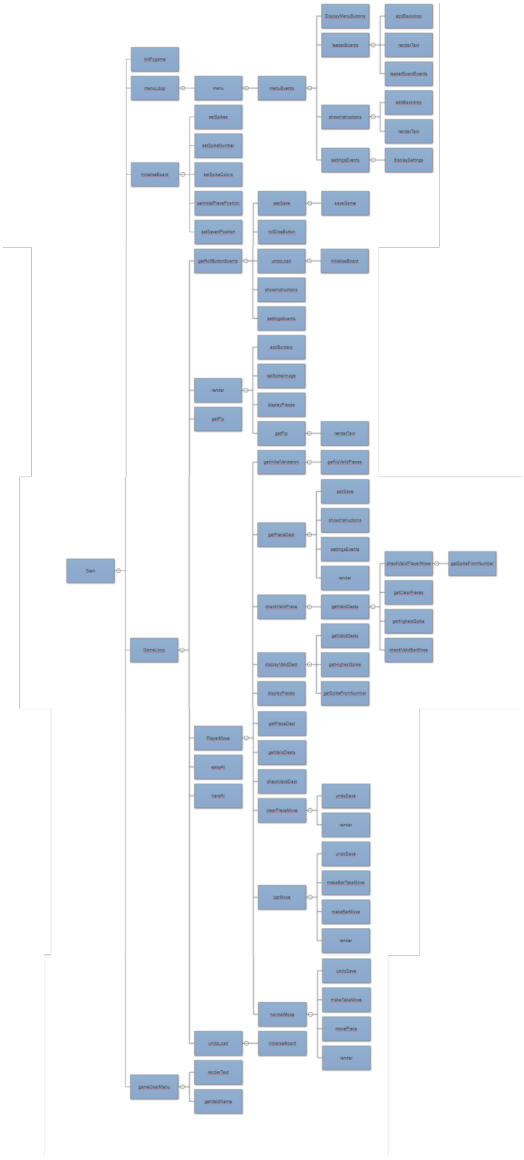


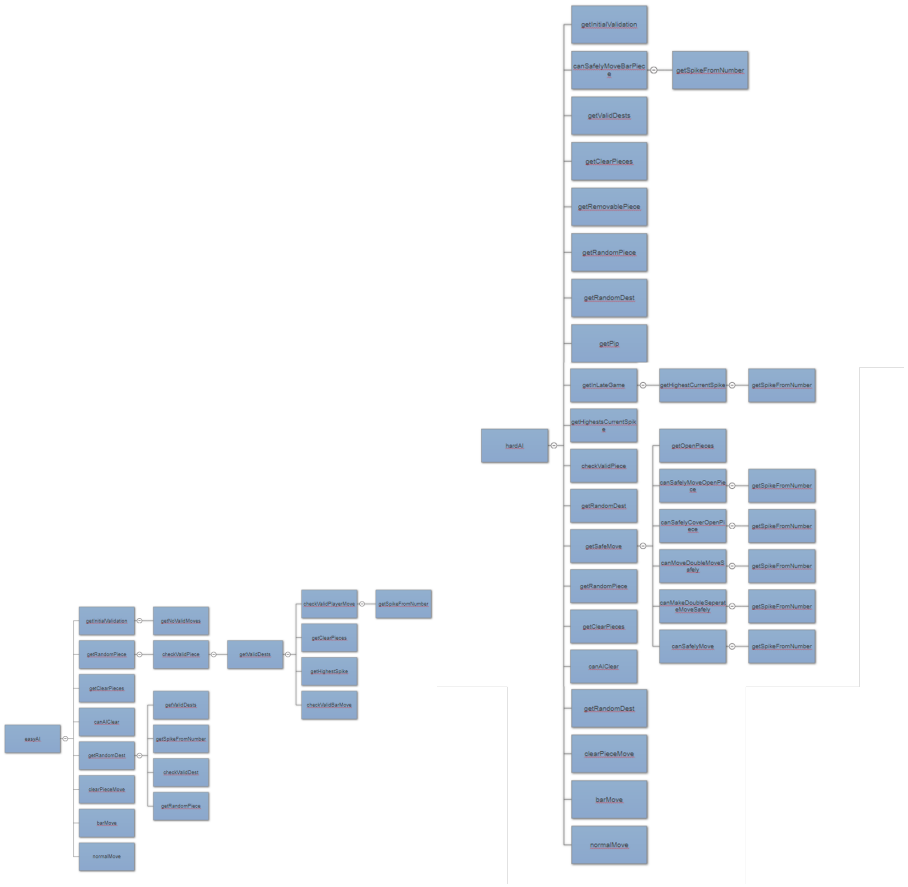
Figure 2.11: Class diagram of my project

2.2.3 Hierarchy Diagrams



(a) Main code excluding AI

Figure 2.12: Hierarchy diagram of functions



(b) Easy AI (c) Hard AI

Figure 2.12: Hierarchy diagram of functions

2.3 Data Structures

Listing 2.3: Stack for storing board states for undoing in the game

```
1  CLASS MyStack
2          DEF constructor()
3                  self.size = 5
4                  self.stack = [0 FOR i=0 TO length(self.size)]
5                  self.front = -1
6                  self.inStack = 0
7          ENDSUB
8
9          DEF push(item)
10                 IF self.front != self.size-1 THEN
11                         IF self.inStack < 5 THEN
12                                 self.inStack = self.inStack + 1
13                         ENDIF
14                         self.front = (self.front + 1) MOD self.size
15                         self.stack[self.front] = item
16                 ELSE
17                         self.inStack = self.inStack - 1
18                         self.front = -1
19                         self.push(item)
20                 ENDIF
21         ENDSUB
22
23         DEF pop()
24                 IF self.isEmpty() == False THEN
25                         self.inStack = self.inStack - 1
26                         self.front = (self.front - 1) MOD self.size
27                         RETURN self.stack[(self.front + 1) MOD self.size]
28                 ELSE
29                         RETURN False
30                 ENDIF
31         ENDSUB
32
33         DEF isEmpty()
34                 RETURN self.inStack == 0
35         ENDSUB
```

The class MyStack is a modified stack. The stack has a maximum size of 5 however the stack can never become "full" as if you add to it when there are 5 items in it, it will overwrite the oldest item. Therefore the stack only holds a maximum of 5 items at a given time but you can always add to it.

Table 2.6: Table of variables in listing 2.3 (page 30)

Variable Name	Data Type	Validation	What it Controls
size	Integer	N/A	Sets the maximum size of the stack
stack	Array	N/A	Stores board states
front	Integer	N/A	Pointer to keep track of the front of the array
inStack	Integer	N/A	Keeps track of number of items in the list

2.4 Storage

Listing 2.4: Reading from CSV file

```
1 SUB loadLeaderboard()
2       lb <- []
3       scores[] <- READ self.save_file
4       FOR score IN scores
5           lb.append(score)
6       ENDFOR
7       END READ
8 ENDSUB
```

Listing 2.4 is a proposed function for loading all information ([firstname, lastname, score]) from the csv file 'highscores.csv' stored in 'self.save_file' into an array 'lb'. From there the leaderboard can be sorted using a merge sort and a radix sort in ascending or descending order.

Listing 2.5: Writing to a CSV file

```
1 SUB appendScore(fn,ln,sc)
2       WRITE self.save_file, fn,ln,score
3 ENDSUB
```

Listing 2.5 is a proposed function for appending information to csv file 'high-scores.csv'. Information to be appended are passed in as parameters and are firstname, lastname and score.

2.5 Preliminary Testing Table

I will be testing my program to ensure that it meets the criteria set out in my objectives (see page 10).

Table 2.7: Table of tests to carry out on my program

Test name	Justification	Test data	Prediction
New game button	Objective 1a	Click on new game button	Game will load as new game
Button hover	Objective 1b	Hover mouse over new game button	The button will become darker
Load game button	Objective 1a	Click on load game button	Game will load from a csv file
Quit button	Objective 1a	Click on quit button	The window will close
Settings button	Objective 1c	Click on settings button	Settings screen will show
Settings back button	Objective 1d	Click on back button in settings	Main menu screen should be shown
Instructions button	Objective 1c	Click on instructions button	Instructions will be displayed over the screen
Leaderboards button	Objective 1c,5b	Click on leaderboards button	Leaderboards screen will be shown
Roll button	Test whether dice are rolled	Click on roll button	Dice will roll randomly
Piece selection	Objective 2a	Click on a piece	Destinations will light up
Undo button	Objective 2bi	Click on undo button	Game should undo 1 turn
Undo button when stack is empty	Objective 2bii	Click on undo button at the start of a game	Error message should appear

Test name	Justification	Test data	Prediction
Settings button in game	Objective 2c	Click on settings button during a game	Settings screen will be shown
Intructions button in game	Objective 2c	Click on instructions button during a game	Instructions will be displayed over the screen
Toggle settings piece colour in game	Objective 2ci	Click on piece colour toggle switch during a game	Pieces in game should change colour
Toggle settings AI difficulty in game	Objective 2cii	Click on AI difficulty toggle switch during a game	Error message should appear
Invalid piece selection I	Objective 3a	Click on board piece when there is a piece taken on the bar	Error message should appear
Invalid piece selection II	Objective 3b	Click on opponents piece	Error message should appear
Destination selection I	Objective 3ci	Select destination with no pieces	Piece should move to destination
Destination selection II	Objective 3cii	Select destination with **1 or more** of your pieces on	Piece should move to destination
Destination selection III	Objective 3ciii	Select destination with **only 1** of the opponents pieces on	Piece should move to destination. Opponents piece should move to bar.
Destination selection IV	Objective 3civ	Select destination as bear off bar when all pieces are in the home quarter	Piece should be removed from the board
Roll a double	Objective 3d	Roll dice (hardcode a double)	4 dice will show up
Taken piece	Test whether pieces can be moved off the bar back into play	Select piece on bar and select a destination	Piece will move off the bar

Test name	Justification	Test data	Prediction
No valid moves	Test what happens when a player rolls and has no valid moves they can make	Roll the dice (hardcode a roll resulting in no valid moves)	Error message should appear. Turn will end.
Easy AI	Objective 4a	Start new game against easy AI. Hardcode a 5-3 roll for the AI	AI will make 2 random moves (likely to leave pieces open)
Hard AI	Objective 4b	Start new game against hard AI. Hardcode a 5-3 roll for the AI	Hard AI should make the famous 5-3 move (see figure 2.13 below), keeping pieces safe.
Hard AI late game strategy	Objective 4b	Start game where all pieces are clear of the others	Hard AI will use late game strategy and prioritise moving pieces furthers back rather than making safe moves
AI taken	Test whether AI can move a piece off the bar	Take AI piece	AI should move piece off bar
AI normal move	Test whether AI can make normal moves	Let AI have a turn	AI should make 2 valid moves
AI bear off move	Test whether AI can bear off pieces	Load game where AI has all pieces in the home quarter	AI should move a piece off the board
AI no valid moves	Test what will happen when the AI cannot make any valid moves	Roll the dice (hardcode a roll resulting in no valid moves)	AI's turn should end. Players turn should begin

Test name	Justification	Test data	Prediction
Exit and save game	Test whether user is prompted with the option to save game before quitting	Exit window during a game	Yes/no box will appear asking whether they wish to save the game
Game over screen	Objective 5a	Complete game	Game over screen should appear prompting the winner to enter their name
Valid name input	Objective 5ai	Enter valid name 'john smith'	Return to main menu. Name will be first in leaderboard under most recent.
Invalid name input I	Objective 5ai	Enter invalid name 'johnsmith'	Error message should appear
Invalid name input II	Objective 5ai	Enter invalid name 'johnsmith '	Error message should appear
Invalid name input III	Objective 5aii	Enter invalid name 'john smith '	Error message should appear
Leaderboard sort recent	Objective 5ci	Complete game. Enter name 'most recent'. Select sort leaderboard by most recent	'most recent' will appear at the top
Leaderboard sort firstname	Objective 5cii	Select sort leaderboard by firstname	Leaderboard should list entries in alphabetical order according to firstname
Leaderboard sort surname	Objective 5ciii	Select sort leaderboard by surname	Leaderboard should list entries in alphabetical order according to surname
Leaderboard sort score	Objective 5civ	Select sort leaderboard by score	Leaderboard should list entries in numerical order according to players winning score

Test name	Justification	Test data	Prediction
Leaderboard order ascending	Objective 5di	Select order leaderboard in ascending order	Leaderboard should list entries in ascending order (e.g. alphabetically from A-Z, by score low-high)
Leaderboard order descending	Objective 5dii	Select order leaderboard in descending order	Leaderboard should list entries in descending order (e.g. alphabetically from Z-A, by score high-low)

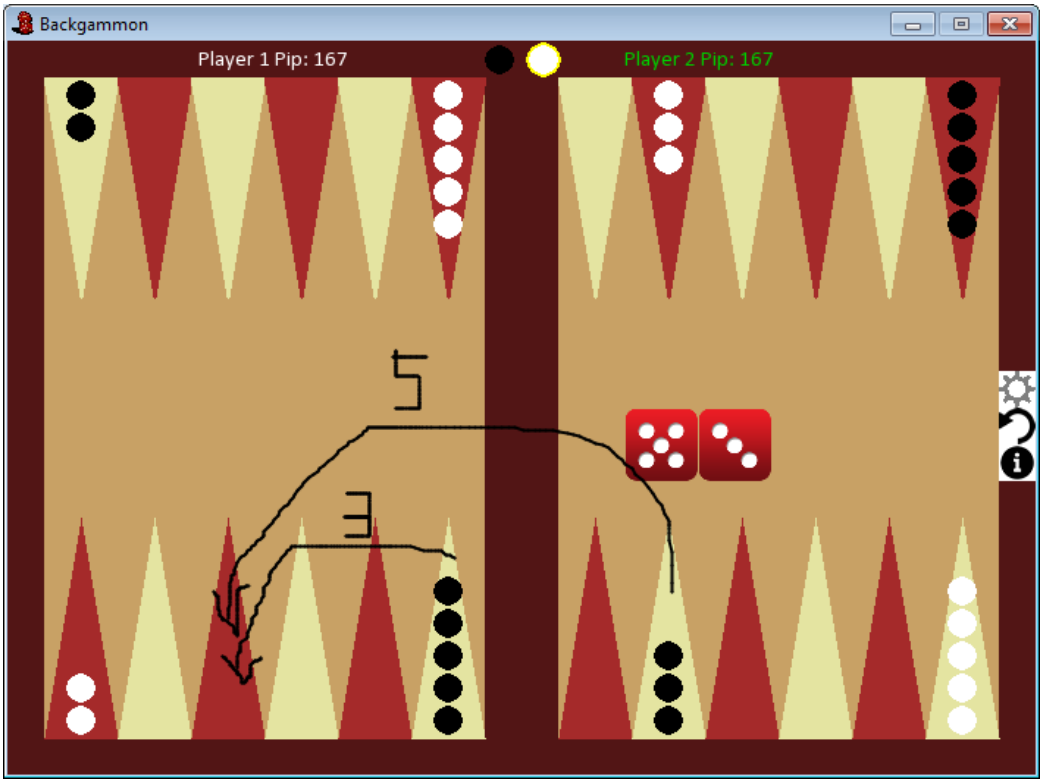


Figure 2.13: Example of the 5-3 starting move

Figure 2.13 is a demonstration of the 5-3 move at the start of a game. It is a classic starting move to ensure no pieces are left open. The move works with other combinations of dice with a difference of 2, for example 4-2, 6-4. I think this will be a good way to test the difference between my easy and hard AI algorithms. As the easy AI makes 2 random moves, it is unlikely that it will play the 5-3 move and will instead leave 2 pieces open which could be taken. However my hard AI checks lots of different moves to see which would be the safest (i.e. not leaving any pieces open). Therefore my hard AI should do the 5-3 move as it aims to reduce the chance of being taken.

3 Technical Solution

3.1 Main Code

Listing 3.1: backgammon.py

```
import pygame,time,random,csv,tkinter,os
import tkinter.messagebox as mb
from classUndoStack import MyStack
from classButton import MyButton
from classGame import MyGame
from classLeaderboard import MyLeaderboard
from classToggle import MyToggleSwitch,MyToggleSwitchColor
import keyboard_input
import hard_ai_two_point_oh as hard_ai

#windows = \\
#linux = //

#Image loading
current_directory = os.getcwd()
RollButtonImg = pygame.image.load("{}//Images//button_roll.png".format(current_directory))
RollButtonHoverImg = pygame.image.load("{}//Images//button_roll_hover.png".format(current_directory))
undoButtonImg = pygame.image.load("{}//Images//undo.png".format(current_directory))

menu_img = pygame.image.load("{}//Images//Menu//menuImg.png".format(current_directory))
newGameImg = pygame.image.load("{}//Images//Menu//new_game.png".format(current_directory))
newGameHoverImg = pygame.image.load("{}//Images//Menu//new_game_hover.png".format(current_directory))
loadGameImg = pygame.image.load("{}//Images//Menu//load_game.png".format(current_directory))
loadGameHoverImg = pygame.image.load("{}//Images//Menu//load_game_hover.png".format(current_directory))
quitImg = pygame.image.load("{}//Images//Menu//quit.png".format(current_directory))
quitHoverImg = pygame.image.load("{}//Images//Menu//quit_hover.png".format(current_directory))

settingsBackground = pygame.image.load("{}//Images//Menu//settings_background.png".format(current_directory))
settingsImgMenu = pygame.image.load("{}//Images//Menu//settings.png".format(current_directory))
settingsImgInGame = pygame.image.load("{}//Images//settings_in_game.png".format(current_directory))
sliderOnImg = pygame.image.load("{}//Images//slider_on.png".format(current_directory))
sliderOffImg = pygame.image.load("{}//Images//slider_off.png".format(current_directory))

leaderboardImg = pygame.image.load("{}//Images//Menu//leaderboard.png".format(current_directory))
leaderboardBackground = pygame.image.load("{}//Images//Menu//leaderboard_background.png".format(current_directory))
instructionsImgMenu = pygame.image.load("{}//Images//Menu//instructions.png".format(current_directory))
instructionsImgInGame = pygame.image.load("{}//Images//instructions_in_game.png".format(current_directory))
gameOverBackground = pygame.image.load("{}//Images//Menu//game_over_screen.png".format(current_directory))

window_icon = pygame.image.load("{}//Images//icon.png".format(current_directory))

#global color variable definitions
maroonBrown = (165,42,42)
beige = (228,228,161)
background = (200,161,101)
borderBrown = (82,21,21)
validGreen = (0,205,0,0.01)
brightRed = (255,0,0)

black = (0,0,0)
white = (255,255,255)
yellow = (255,255,0)
```

```
53 grey = (211,211,211)
54
55 #Global variable definitions
56 PlayerColor = "Black"
57 AIColor = "White"
58 SAVEDGAME = "saved_game.csv" #name of csv file to load
59 AIPAUSE = 1   #length to pause between ai moves
60 size = width, height = 700,500
61
62 #Classes
63 rollButton = MyButton((size[0]/4)*2.25, size[1]/2, RollButtonImg)
64 Quit = MyButton((size[0]/2-int(quitImg.get_width()/2)),400,quitImg)
65 new = MyButton((size[0]/2-int(newGameImg.get_width()/2)),250,newGameImg)
66 load = MyButton((size[0]/2-int(loadGameImg.get_width()/2)),325,loadGameImg)
67 leaderboardButton = MyButton(size[0]-100, size[1]-75, leaderboardImg)
68 instructionsButtonMenu = MyButton(size[0]-75, size[1]-125, instructionsImgMenu)
69 instructionsButtonInGame = MyButton(size[0]-25, size[1]/2+25, instructionsImgInGame)
70 undoButton = MyButton(size[0]-25, size[1]/2, undoButtonImg)
71 settingsButtonMenu = MyButton(size[0]-75, size[1]-185, settingsImgMenu)
72 settingsButtonInGame = MyButton(size[0]-25, size[1]/2-25, settingsImgInGame)
73
74 #toggle switches
75 toggleHighlightMoves = MyToggleSwitch(sliderOnImg, sliderOffImg, 170, 150, 'Highlight Moves
       On', 'Highlight Moves Off')
76 toggleAIDifficulty = MyToggleSwitch(sliderOnImg, sliderOffImg, 170, 250, 'Hard AI', 'Easy AI'
       )
77 togglePieceColor = MyToggleSwitchColor(sliderOnImg, sliderOffImg, 450, 150, 'Black vs White'
       , 'Blue vs Red')
78 toggleDiceAnimation = MyToggleSwitch(sliderOnImg, sliderOffImg, 450, 250, 'Dice Animation On
       ', 'Dice Animation Off')
79 toggle12Player = MyToggleSwitch(sliderOnImg, sliderOffImg, 310, 350, '1 Player', '2 Player')
80
81 toggleAIDifficulty = MyToggleSwitch(sliderOnImg, sliderOffImg, 170, 150, 'Hard AI', 'Easy AI'
       )
82 toggle12Player = MyToggleSwitch(sliderOnImg, sliderOffImg, 450, 150, '1 Player', '2 Player')
83 toggleHighlightMoves = MyToggleSwitch(sliderOnImg, sliderOffImg, 170, 250, 'Highlight Moves
       On', 'Highlight Moves Off')
84 toggleDiceAnimation = MyToggleSwitch(sliderOnImg, sliderOffImg, 450, 250, 'Dice Animation On
       ', 'Dice Animation Off')
85 togglePieceColor = MyToggleSwitchColor(sliderOnImg, sliderOffImg, 170, 350, 'Black vs White'
       , 'Blue vs Red')
86 toggleAIPause = MyToggleSwitch(sliderOnImg, sliderOffImg, 450, 350, 'AI Pause On', 'AI Pause
       Off')
87
88 class Spike(): #class spike for each spike on board
89     def __init__(self):
90         self.color = None
91         self.containsColor = None
92         self.NumberOfPieces = 0
93         self.image = None
94         self.spikeNo = None
95         self.baseX = None
96         self.baseY = None
97         self.lcoord = None
98         self.rcoord = None
99         self.centercoord = None
100
101    def setColor(self, color):  #Set the physical colour of the spike
102        self.color = color
103
104    def setContainsColor(self, color): #Set the colour of pieces that the spike contains
105        self.containsColor = color
106
107    def setImage(self, left, right, center):
108        self.image = pygame.draw.polygon(screen, self.getColor(), ((left),(right),(
       center)), 0)
109
110    def setNumberOfPieces(self,n): #Increase/decrease the number of pieces on the spike
111        self.NumberOfPieces += n
112        if self.NumberOfPieces == 0:    #If there are now no pieces
113            self.setContainsColor(None)     #set contains colour of no pieces
114
115    def setSpikeNo(self,i):
116        i = int(i)
117        self.spikeNo = i
118
119    def setBaseX(self,x):
```

```
120         self.baseX = x
121
122     def setBaseY(self,y):
123         self.baseY = y
124
125     def setLeftCoord(self,coord):
126         self.lcoord = coord
127
128     def setRightCoord(self,coord):
129         self.rcoord = coord
130
131     def setCenterCoord(self,coord):
132         self.centercoord = coord
133
134     def getColor(self):
135         return (self.color[0],self.color[1],self.color[2])
136     def getContainsColor(self):
137         return self.containsColor
138     def getImage(self):
139         return self.image
140     def getNumberOfPieces(self):
141         return self.NumberOfPieces
142     def getSpikeNo(self):
143         return self.spikeNo
144     def getBaseX(self):
145         return self.baseX
146     def getBaseY(self):
147         return self.baseY
148     def getLeftCoord(self):
149         return self.lcoord
150     def getRightCoord(self):
151         return self.rcoord
152     def getCenterCoord(self):
153         return self.centercoord
154
155 class MyBar():
156     def __init__(self):
157         self.pieces = []
158         self.x = 25+(50*6)
159         self.y = 25
160         self.width = 50
161         self.height = size[1]-(25*2)
162         self.img = pygame.draw.rect(screen,borderBrown,(self.x,self.y,self.width,self.
        height),0)
163
164     def set_piece(self,color): #add a piece to the bar
165         self.pieces.append(color)
166
167     def remove_piece(self,color):   #remove a piece from the bar
168         self.pieces.remove(color)
169
170     def displayBarRect(self):
171         pygame.draw.rect(screen,borderBrown,(self.x,self.y,self.width,self.height),0)
172
173     def getNoPieces(self,color):
174         n = 0
175         for piece in self.pieces:
176             if piece == color:
177                 n+=1
178         return n
179
180     def getPieceList(self):
181         return self.pieces
182
183     def getBarImg(self):
184         return self.img
185
186 class MyDice():
187     def __init__(self):
188         self.rollList = []
189         self.diceImages = self.loadDiceImages()
190     def loadDiceImages(self):
191         dice = [0,0,0,0,0,0] #array containing 6 dice images
192         for i in range(6):
193             n = pygame.image.load("{}//Images//dice-{}.png".format(current_directory,i
        +1)) #load image of each dice from directory
194             n = pygame.transform.scale(n,(50,50))       #transform the image smaller
```

```
195             dice[i] = n                                     #place dice image into list
196         return dice
197
198     def roll_dice_effect(self):
199         for i in range(10):                 #10 times
200             num1 = random.randint(1,6)  #get a random number
201             pos1 = num1 - 1                 # get the dice image position in the list
202             screen.blit(self.diceImages[pos1],((size[0]/4)*2.8,size[1]/2))  #display the
        dice
203
204             num2 = random.randint(1,6) #same as above for the other dice
205             pos2 = num2 - 1
206             screen.blit(self.diceImages[pos2],((size[0]/4)*3.2,size[1]/2))
207             pygame.display.update()
208             time.sleep(.1)
209
210     def roll_dice(self):
211         if toggleDiceAnimation.getStatus() == True: #if dice animation is on
212             self.roll_dice_effect()             #show rolling dice effect
213         num1 = random.randint(1,6)      #pick random number between 1 and 6
214         num2 = random.randint(1,6)
215         pos1 = num1 - 1                 #position is for the dice image to display since
        index starts at 0
216         pos2 = num2 - 1
217         if num1 == num2:                #if the numbers are the same
218             self.rollList = [num1,num1,num1,num1]   #add 4 of the number to the roll
       list
219         else:
220             self.rollList = [num1,num2]
221
222     def displayDice(self):
223         if len(self.rollList) == 0:             #if there are no numbers in the roll
       list
224             screen.blit(rollButton.getImage(),(rollButton.getX(),rollButton.getY()))
225         else:
226             distance = 0
227             for i in range(len(self.rollList)): #for each number in the roll list
228                 screen.blit(self.diceImages[self.rollList[i]-1],(((size[0]/4)*2.4)+
       distance,size[1]/2)) #display the dice image
229                 distance+=50
230     def setRollList(self,myList):
231         self.rollList = myList
232     def getRollList(self):
233         return self.rollList
234     def removeNumber(self,n):
235         self.rollList.remove(n)
236 #
       ----------------------------------------------------SPIKES-----------------------------------------------------
       #
237
238 def setSpikes(): #set every item in 2d array board as a spike class
239     for i in range(2):
240         for k in range(12):
241             Board[i][k] = Spike()
242
243 def setSpikeNumber():   #set a number for each spike from 1 to 24 going round the board
       clockwise
244     for i in range(0,12):
245         Board[0][i].setSpikeNo(i+1)
246         Board[1][i].setSpikeNo(24-i)
247
248 def setSpikeColors(): #set alternate colors for each spike
249     for i in range(0,12,2):         #every other spike to set colour on both rows
250         Board[0][i].setColor(beige)
251         Board[1][i].setColor(maroonBrown)
252
253     for i in range(1,13,2):         #every other spike to set colour on both rows
254         Board[0][i].setColor(maroonBrown)
255         Board[1][i].setColor(beige)
256
257 def setSpikeImage():    #set each spikes triangle image
258     y1 = 25
259     x1 = 25
260     for i in range(12):         #every other spike to set colour on both rows
261         if i == 6:
262             x1+=50
263         Board[0][i].setImage((x1,y1),(x1+50,y1),(x1+25,y1+150)) #draws spike
```

```
264         Board[0][i].setBaseX(x1+25)                                    #spikes center x
      coordinate for moving piece to spike
265         Board[0][i].setBaseY(y1+12)                                      #spikes y
      coordinate at base for stacking pieces
266         Board[0][i].setLeftCoord((x1,y1))                                  #this and
      next 2 lines set coordinates for drawing green triangle over it for valid moves
267         Board[0][i].setRightCoord((x1+50,y1))
268         Board[0][i].setCenterCoord((x1+25,y1+150))
269
270         Board[1][i].setImage((x1,size[1]-y1),(x1+50,size[1]-y1),(x1+25,size[1]-y1-150))
271         Board[1][i].setBaseX(x1+25)
272         Board[1][i].setBaseY(size[1]-y1-12)
273         Board[1][i].setLeftCoord((x1,size[1]-y1))
274         Board[1][i].setRightCoord((x1+50,size[1]-y1))
275         Board[1][i].setCenterCoord((x1+25,size[1]-y1-150))
276
277         x1+=50
278
279 def getSpikeFromNumber(target_no):
280     i = 0
281     k = 0
282     while i < 2: #iterate through all spikes on the board
283         while k < 12:
284             if Board[i][k].getSpikeNo() == target_no: #if the spikes number is the
      number you want to find
285                 return Board[i][k]      #return the spike class
286             k+=1
287         i+=1
288         k=0
289
290 def getHighestSpike(color): #find the spike with the highest number containing pieces of
       a colour
291     if color == Game.PlayerColor:
292         boardNumber = 1
293     else:
294         boardNumber = 0
295     count = 5
296     while count > -1:
297         piece = Board[boardNumber][count]
298         if piece.getNumberOfPieces() > 0 and piece.getContainsColor()==color:
299             return piece
300         count -=1
301 #
      ----------------------------------------------PIECES-----------------------------------------------------
      #
302
303 def setInitialPiecePosition():
304     #black pieces moving clockwise
305     Board[0][0].setNumberOfPieces(2)
306     Board[0][11].setNumberOfPieces(5)
307     Board[1][7].setNumberOfPieces(3)
308     Board[1][5].setNumberOfPieces(5)
309
310     #set spike contain colour to black
311     Board[0][0].setContainsColor(Game.PlayerColor)
312     Board[0][11].setContainsColor(Game.PlayerColor)
313     Board[1][7].setContainsColor(Game.PlayerColor)
314     Board[1][5].setContainsColor(Game.PlayerColor)
315
316     #white pieces moving anit-clockwise
317     Board[1][0].setNumberOfPieces(2)
318     Board[1][11].setNumberOfPieces(5)
319     Board[0][7].setNumberOfPieces(3)
320     Board[0][5].setNumberOfPieces(5)
321
322     #set spike contain colour to white
323     Board[1][0].setContainsColor(Game.AIColor)
324     Board[1][11].setContainsColor(Game.AIColor)
325     Board[0][7].setContainsColor(Game.AIColor)
326     Board[0][5].setContainsColor(Game.AIColor)
327
328 def displayPieces():
329     for i in range(12):
330         if Board[0][i].getNumberOfPieces() != 0: #if the spike has pieces on
331             number_of_pieces = Board[0][i].getNumberOfPieces()  #get the number of
      pieces it has
332             pieceDistance = 0
```

```
333             count = 0
334             while count < number_of_pieces and count < 6: #draw up to 6 pieces
335                 if Board[0][i].getContainsColor() == Game.PlayerColor: #determine the
        color of piece to draw
336                     color = togglePieceColor.getPieceColor()[0]
337                 else: color = togglePieceColor.getPieceColor()[1]
338                 pygame.draw.circle(screen,color,(Board[0][i].getBaseX(),Board[0][i].
        getBaseY()+pieceDistance),10,0) #draw the piece
339                 pieceDistance+=22   #increase height
340                 count+=1
341
342             number_of_pieces -= count
343             if number_of_pieces != 0: #if there are more than 6 pieces
344                 if color == togglePieceColor.getPieceColor()[0]:
345                     text_color = togglePieceColor.getPieceColor()[1]
346                     #get opposite color to piece color for text
347                 else:
348                     text_color = togglePieceColor.getPieceColor()[0]
349                 renderText(str(number_of_pieces),15,text_color,Board[0][i].getBaseX()-3,
        Board[0][i].getBaseY()+pieceDistance-29)
350                 #text showing number of additional pieces
351
352
353         if Board[1][i].getNumberOfPieces() != 0:
354             number_of_pieces = Board[1][i].getNumberOfPieces()
355             pieceDistance = 0
356             count = 0
357             while count < number_of_pieces and count < 6:
358                 if Board[1][i].getContainsColor() == Game.PlayerColor:
359                     color = togglePieceColor.getPieceColor()[0]
360                 else: color = togglePieceColor.getPieceColor()[1]
361                 pygame.draw.circle(screen,color,(Board[1][i].getBaseX(),Board[1][i].
        getBaseY()-pieceDistance),10,0)
362                 pieceDistance+=22
363                 count+=1
364
365             number_of_pieces -= count
366             if number_of_pieces != 0:
367                 if color == togglePieceColor.getPieceColor()[0]: text_color =
        togglePieceColor.getPieceColor()[1]
368                 else: text_color = togglePieceColor.getPieceColor()[0]
369                 renderText(str(number_of_pieces),15,text_color,Board[1][i].getBaseX()-3,
        Board[1][i].getBaseY()-pieceDistance+15)
370     #bar piece displaying
371     pieceDistance = 0
372     for piece in Bar.getPieceList():
373         if piece == Game.PlayerColor:
374             color = togglePieceColor.getPieceColor()[0]
375         else:
376             color = togglePieceColor.getPieceColor()[1]
377         pygame.draw.circle(screen,color,(int(round(size[0]/2)),int(round(size[1]/2 +
        pieceDistance))),10,0)
378         pieceDistance += 22
379 #
        ----------------------------------------------BOARD----------------------------------------------
        #
380
381 def addBorders(): #adds boarders around board
382     topHorz = pygame.draw.rect(screen,borderBrown,(0,0,size[0],25),0)
383     bottomHorz = pygame.draw.rect(screen,borderBrown,(0,size[1]-25,size[0],25),0)
384     rightVert = pygame.draw.rect(screen,borderBrown,(size[0]-25,25,25,size[1]-50),0)
385     leftVert = pygame.draw.rect(screen,borderBrown,(0,25,25,size[1]-50),0)
386     buttonsBar = pygame.draw.rect(screen,white,(size[0]-25,(size[1]/2),25,50),0)
387
388     #adding background square for in game buttons to show hovering
389     mx, my = pygame.mouse.get_pos()
390     if undoButton.getWithin(mx,my): undo_bg_color = grey     #if the mouse coordinates
        are over the button, make square grey to show hover
391     else: undo_bg_color = white                              #otherwise show white to
        show no hover
392     if instructionsButtonInGame.getWithin(mx,my): instructions_bg_color = grey
393     else: instructions_bg_color = white
394     if settingsButtonInGame.getWithin(mx,my): settings_bg_color = grey
395     else: settings_bg_color = white
396     undoBg = pygame.draw.rect(screen,undo_bg_color,(size[0]-25,(size[1]/2),25,25),0) #
        draw the background square
397     instructionBg = pygame.draw.rect(screen,instructions_bg_color,(size[0]-25,(size
```

```
      [1]/2)+25,25,25),0)
398       settingsBg = pygame.draw.rect(screen,settings_bg_color,(size[0]-25,(size[1]/2)
      -25,25,25),0)
399
400   def RollDiceButton(mx,my):
401       if rollButton.getWithin(mx,my): #if mouse coordiantes are over the roll dice button
402           if rollButton.getImage() != RollButtonHoverImg: #if the roll button image isn't
      the hover image (to prevent unnecessary assignment of new image)
403               rollButton.setImage(RollButtonHoverImg)          #set the image to the hover
      image
404       else:
405           if rollButton.getImage() != RollButtonImg: rollButton.setImage(RollButtonImg) #
      if the image isn't normal image set to normal image
406       screen.blit(rollButton.getImage(),(rollButton.getX(),rollButton.getY()))         #
      show the image
407
408   def getRollButtonEvents():
409       for event in pygame.event.get(): #event to allow player to quit
410           if event.type == pygame.QUIT:
411               askSave() #save game
412
413           mx, my = pygame.mouse.get_pos()
414           RollDiceButton(mx,my)
415           if event.type == pygame.MOUSEBUTTONDOWN:
416               if rollButton.getWithin(mx,my): #if click on roll button
417                   Dice.roll_dice() #roll the dice
418                   return True
419               elif undoButton.getWithin(mx,my): #if click the undo button
420                   undoLoad() #perform an undo
421                   return True
422               elif instructionsButtonInGame.getWithin(mx,my): showInstructions() #if click
       instructions, show the instructions
423               elif settingsButtonInGame.getWithin(mx,my): settingsEvents() #if click
      settings, show the settings screen
424       return False
425   #
      ------------------------------------------------------INITIALISE-------------------------------------------------------
      #
426
427   def initialiseBoard(game_type):
428       setSpikes()
429       setSpikeNumber()
430       setSpikeColors()
431
432       if game_type == 'new':
433           setInitialPiecePosition() #set pieces in starting position
434           Game.setTurn(Game.PlayerColor) #start game as blacks turn
435       elif game_type == 'load':
436           color = setSavedPosition(SAVEDGAME) #set piece position saved from csv file
437       elif game_type == 'empty':
438           Game.setTurn(None)
439
440   def initPygame():
441       pygame.init()
442       pygame.display.set_caption('Backgammon')
443       pygame.display.set_icon(window_icon)
444       screen = pygame.display.set_mode(size)
445       clock = pygame.time.Clock()
446       return screen,clock
447
448   def setSavedPosition(file_name):
449       try:
450           saved_file = open("{}//CSV//{}".format(current_directory,file_name)) #open csv
      file
451           saved = csv.reader(saved_file,delimiter=',')
452           save_list = []
453           for row in saved:
454               save_list.append(row) #copy contents to save_list
455
456           count = 0
457           for i in range(2): #iterate through the board
458               for k in range(12):
459                   Board[i][k].setNumberOfPieces(int(save_list[count][0]))
460                   #set number of pieces
461                   if save_list[count][1] == 'b': #set contains black
462                       set_color = Game.PlayerColor
463                   elif save_list[count][1] == 'w': #set contains white
```

```
464                     set_color = Game.AIColor
465                 else:       #set contains none
466                     set_color = None
467                 Board[i][k].setContainsColor(set_color)
468                 count+=1
469         for black in range(int(save_list[24][0])): #add bar pieces
470             Bar.set_piece(Game.PlayerColor)
471         for white in range(int(save_list[24][1])):
472             Bar.set_piece(Game.AIColor)
473 
474         while "" in save_list[26] == True:
475             save_list[26].remove("")
476         colorGo = "".join(save_list[26]) #set whose turn it is
477         Game.setBothMoves(int(save_list[27][0]),int(save_list[27][1]))
478         #set the number of moves both players had already made
479 
480         try:
481             tempRollList = save_list[25]
482             new_dice_list = []
483             for number in tempRollList:
484                 if number != '':
485                     new_dice_list.append(int(number))
486             Dice.setRollList(new_dice_list) #load dice rolled from file
487         except:
488             Dice.setRollList([]) #set empty if error
489         finally:
490             Game.setTurn(colorGo)
491     except:
492         #error if unable to open and read file
493         #set game to new
494         tkinterMessageBox(1,"Error","Unable to load game. Starting new one.")
495         initialiseBoard('new')
496         Game.setTurn(Game.PlayerColor)
497 #-----------------------------------------------SAVING
      GAME-----------------------------------------------------------#
498 
499 def saveGame(file_name): #save game to csv file
500     try:
501         save_file = open("{}//CSV//{}".format(current_directory,file_name),'w',newline='
      ')
502         csvwriter = csv.writer(save_file,delimiter =',')
503         for i in range(2):
504             for k in range(12):
505                 number_of_pieces = Board[i][k].getNumberOfPieces()
506                 color = Board[i][k].getContainsColor()
507                 if color == Game.PlayerColor:color='b'
508                 elif color == Game.AIColor:color='w'
509                 else:color = "None"
510                 rowwrite = number_of_pieces,color
511                 csvwriter.writerow(rowwrite)
512         black_bar = Bar.getNoPieces(Game.PlayerColor)
513         white_bar = Bar.getNoPieces(Game.AIColor)
514         csvwriter.writerow([black_bar,white_bar])
515         csvwriter.writerow(list(Dice.getRollList()))
516         csvwriter.writerow(Game.getTurn())
517         csvwriter.writerow(Game.getBothMoves())
518         save_file.close()
519     except:
520         save_file.close()
521         tkinterMessageBox(1,"Error","Unable to save game")
522         pygame.quit()
523         exit()
524 
525 def askSave(): #ask user if they wish to save the game
526     save = tkinterMessageBox(2,"Warning","Would you like to save your game?")
527 
528     if save:
529         saveGame("saved_game.csv")
530     pygame.quit()
531     exit()
532 
533 def undoSave(): #save the game to undo stack (temporary save)
534     tempBoard = []
535 
536     #iterate through the board and copy all data to tempBoard
537     for i in range(2):
538         for k in range(12):
```

```
539                 number_of_pieces = Board[i][k].getNumberOfPieces()
540                 color = Board[i][k].getContainsColor()
541                 if color == Game.PlayerColor:color='b'
542                 elif color == Game.AIColor:color='w'
543                 else:color = "None"
544                 tempBoard.append([number_of_pieces,color])
545     bar_black = Bar.getNoPieces(Game.PlayerColor)
546     bar_white = Bar.getNoPieces(Game.AIColor)
547     tempBoard.append([bar_black,bar_white])
548     tempBoard.append(list(Dice.getRollList()))
549     tempBoard.append(Game.getTurn())
550     tempBoard.append(Game.getBothMoves())
551     #push tempBoard to stack
552     undo.push(tempBoard)
553 
554 def undoLoad(): #load game from the undo stack
555     tempBoard = undo.pop() #pop most recent board from stack
556 
557     #copy contents from tempBoard into the board
558     if tempBoard != False:
559         initialiseBoard('empty')
560         Bar.pieces = []
561         save_list = []
562         for row in range(24):
563             #row[0] number of pieces
564             #row[1] color
565             contains = int(tempBoard[row][0])
566             if tempBoard[row][1] == 'b':
567                 set_color = Game.PlayerColor
568             elif tempBoard[row][1] == 'w':
569                 set_color = Game.AIColor
570             else:
571                 set_color = None
572             save_list.append([contains,set_color])
573 
574         count = 0
575         for i in range(2):
576             for k in range(12):
577                 Board[i][k].setNumberOfPieces(save_list[count][0])
578                 Board[i][k].setContainsColor(save_list[count][1])
579                 count+=1
580 
581         for i in range(tempBoard[24][0]):
582             Bar.set_piece(Game.PlayerColor)
583         for k in range(tempBoard[24][1]):
584             Bar.set_piece(Game.AIColor)
585         rolls_left = tempBoard[25]
586         Dice.setRollList(rolls_left)
587         color_turn = tempBoard[26]
588         Game.setTurn(color_turn)
589         Game.setBothMoves(int(tempBoard[27][0]),int(tempBoard[27][1]))
590     else:
591         tkinterMessageBox(1,"Error","You can only undo 5 times. Please make some moves
        first.")
592 #
        ----------------------------------------------------RENDER----------------------------------------------------
        #
593 
594 def renderText(text,font_size,color,x,y):
595     #render text pass variables: text, font size, color and x,y coordinates to blit to
596     font = pygame.font.Font("{}//Fonts//calibri.ttf".format(current_directory),
        font_size)
597     text = screen.blit((font.render(text, 1, color)),(x,y))
598     return text
599 
600 def render():
601     screen.fill(background)
602     addBorders()
603     Bar.displayBarRect()
604     setSpikeImage()
605     displayPieces()
606     screen.blit(undoButton.getImage(),(undoButton.getX(),undoButton.getY())) #display
        undo button
607     screen.blit(instructionsButtonInGame.getImage(),(instructionsButtonInGame.getX(),
        instructionsButtonInGame.getY())) #display instructions button
608     screen.blit(settingsButtonInGame.getImage(),(settingsButtonInGame.getX(),
        settingsButtonInGame.getY())) #display settings button
```

```
609     Dice.displayDice() #show the dice
610     getPip()
611     pygame.display.update()
612
613 def tkinterMessageBox(box_type,title,description):
614     root = tkinter.Tk()
615     root.withdraw()
616     if box_type == 1:
617         response = mb.showerror(title,description) #show an error box
618     elif box_type == 2:
619         response = mb.askyesno(title,description) #show a yes/no question box
620     root.update()
621     return response
622 #----------------------------------------------------GAME
        FUNCTIONS----------------------------------------------------#
623     #------------------VALIDATION--------------#
624 def checkValidBarMove(dest): #check if bar move is valid
625     if Game.getTurn() == Game.PlayerColor:
626         spike_target = dest.getSpikeNo()
627     else: spike_target = 25 - dest.getSpikeNo()
628
629     if spike_target in Dice.getRollList():
630         if (dest.getNumberOfPieces() == 0) or (dest.getNumberOfPieces() == 1 and dest.
        getContainsColor() != Game.getTurn()) or (dest.getContainsColor() == Game.getTurn())
        :
631             return True
632     return False
633
634 def checkValidPlayerMove(piece,target): #check if player move is valid
635         current_color,opposite_color = Game.getColorOpposite()
636         if current_color == Game.PlayerColor:
637             target_no = piece.getSpikeNo()+target #get the spike number of the
        destination
638             if target_no <= piece.getSpikeNo() or target_no > 24: #return false if the
        spike number is less than the pieces spike number because can't move backwards
639                 return False
640         else:
641             target_no = piece.getSpikeNo()-target
642             if target_no >= piece.getSpikeNo() or target_no < 1: #return false if the
        spike number is less than the pieces spike number because can't move backwards
643                 return False
644         target_spike = getSpikeFromNumber(target_no)
645         if (target_spike.getContainsColor() in [current_color,None]) or (target_spike.
        getContainsColor() == opposite_color and target_spike.getNumberOfPieces() == 1):
646             return True
647         else: return False
648
649 def getPip(): #get pip scores for both players
650     blackPip = 0
651     whitePip = 0
652     for i in range(2):
653         for k in range(12):
654             if Board[i][k].getContainsColor() == Game.PlayerColor:
655                 for piece in range(Board[i][k].getNumberOfPieces()):
656                     blackPip += (25 - Board[i][k].getSpikeNo())
657             elif Board[i][k].getContainsColor() == Game.AIColor:
658                 for piece in range(Board[i][k].getNumberOfPieces()):
659                     whitePip += ((0 - Board[i][k].getSpikeNo())*-1)
660
661     blackPip += 25*Bar.getNoPieces(Game.PlayerColor)
662     whitePip += 25*Bar.getNoPieces(Game.AIColor)
663
664     p1Text = "Player 1 Pip: {}".format(blackPip)
665     p2Text = "Player 2 Pip: {}".format(whitePip)
666     if Game.getTurn() == Game.PlayerColor:
667         c1,c2 = validGreen,white
668     else: c1,c2 = white,validGreen
669     renderText(p1Text,15,c1,130,5)
670     renderText(p2Text,15,c2,420,5)
671
672     #show what piece colours turn it is
673     if Game.getTurn() == Game.PlayerColor:
674         pygame.draw.circle(screen,yellow,(335,13),12,0)
675     else: pygame.draw.circle(screen,yellow,(365,13),12,0)
676     pygame.draw.circle(screen,black,(335,13),10,0)
677     pygame.draw.circle(screen,white,(365,13),10,0)
678     return blackPip,whitePip
```

```
679
680 def getClearPieces(color):
681     #determine if players pieces are all in the take off quarter
682     if Bar.getNoPieces(color) > 0:
683         return False
684     if color == Game.PlayerColor:
685         board_1,board_2 = 0,1
686     else:
687         board_1,board_2 = 1,0
688     i = 0
689     while i < len(Board[board_1]):
690         if Board[board_1][i].getContainsColor() == color:
691             if Board[board_1][i].getNumberOfPieces() > 0:
692                 return False
693         i+=1
694     i = 6
695     while i < 12:
696         if Board[board_2][i].getContainsColor() == color:
697             if Board[board_2][i].getNumberOfPieces() > 0:
698                 return False
699         i+=1
700     return True
701
702 def getPieceDest(selected):
703     #get piece/dest selection from user input
704     for event in pygame.event.get():
705         if event.type == pygame.QUIT:
706             askSave()
707
708         mx, my = pygame.mouse.get_pos()
709         if event.type == pygame.MOUSEBUTTONDOWN:
710             i = 0
711             k = 0
712             #iterate through the board
713             while i < 2:
714                 while k < 12:
715                     #if mouse coordinates collide with a spikes image
716                     if Board[i][k].getImage().collidepoint(mx,my):
717                         selected = True
718                         #return this spike as selected
719                         return Board[i][k], selected
720                     else:
721                         k+=1
722                 i+=1
723                 k=0
724             if Bar.getBarImg().collidepoint(mx,my): #if mouse coordinates collide with
    bar rectangle
725                 n = Bar.getNoPieces(Game.getTurn())
726                 if n != 0:
727                     selected = True
728                     return "bar",selected
729             elif mx >= 0 and mx <= 25: #selecting take off var
730                 if my >= size[1]/2 and my <= size[1]-25 or my >= 25 and my <= size[1]/2:
731                     return "takeoff",True
732             elif undoButton.getWithin(mx,my):  #select undo button
733                 return False, False
734             elif instructionsButtonInGame.getWithin(mx,my): #select instructions button
735                 showInstructions()
736                 render()
737             elif settingsButtonInGame.getWithin(mx,my):  #select settings button
738                 settingsEvents()
739                 render()
740     return None,selected
741
742 def getNoValidMoves(): #determine the number of different moves the player could make
743     valid_moves = 0
744     valid_moves+=len(getValidDests('bar'))
745     if valid_moves == 0:
746         for i in range(2):
747             for k in range(12):
748                 if Board[i][k].getContainsColor() == Game.getTurn():
749                     valid_moves += len(getValidDests(Board[i][k]))
750     return valid_moves
751
752 def getValidDests(piece):
753     #get list of all destinations that could be moved to from a piece
754     valid_dests = []
```

```
755     if piece != "bar":
756         for number in Dice.getRollList():
757             valid = checkValidPlayerMove(piece,number) #check if using the dice number gives a valid move
758             if valid:
759                 if Game.getTurn() == Game.PlayerColor:
760                     valid_dests.append(piece.getSpikeNo()+number)  #add to valid_dests if valid
761                 else:
762                     valid_dests.append(piece.getSpikeNo()-number)
763 
764         n = len(Dice.getRollList())
765         count = 0
766         if getClearPieces(Game.getTurn()): #if all pieces are in home quarter
767             while count < n:
768                 if Game.getTurn() == Game.PlayerColor:
769                     if piece.getNumberOfPieces() > 0 and (piece.getSpikeNo() + Dice.getRollList()[count]) == 25:
770                         #if the spike can move with the number off the board - e.g. roll a 6 and have pieces on 6
771                         valid_dests.append("takeoff") #add takeoff to valid_dests
772                         break
773                     else:
774                         highest = getHighestSpike(Game.getTurn())   #get the furthest back spike with your pieces on
775                         if highest.getSpikeNo()+Dice.getRollList()[count] > 25 and piece == highest:
776                             #if furthest back spike + dice number is greater than 25 then valid
777                             #e.g. have pieces on spike 5 but not 6. Roll a 6. Can clear from 5.
778                             valid_dests.append("takeoff")
779                             break
780                 #same for AI
781                 elif Game.getTurn() == Game.AIColor:
782                     if piece.getNumberOfPieces() > 0 and (piece.getSpikeNo() - Dice.getRollList()[count]) == 0:
783                         valid_dests.append("takeoff")
784                         break
785                     else:
786                         highest = getHighestSpike(Game.getTurn())
787                         if highest.getSpikeNo()-Dice.getRollList()[count] < 0 and piece == highest:
788                             valid_dests.append("takeoff")
789                             break
790                 count+=1
791     #get valid moves from the bar
792     elif piece == "bar":
793         for i in range(Bar.getNoPieces(Game.getTurn())):
794             for n in Dice.getRollList():
795                 if Game.getTurn() == Game.PlayerColor:
796                     dest = Board[0][n-1]
797                 else:
798                     dest = Board[1][n-1]
799                 val = checkValidBarMove(dest)
800                 if val:
801                     valid_dests.append(dest.getSpikeNo())
802     return valid_dests
803 
804 def displayValidDest(piece): #show valid destinations as a green spike to signify a valid move
805     valid_dest = getValidDests(piece) #get all the valid dests
806     if "takeoff" in valid_dest: #display bear off bar as green if you can takeoff
807         if Game.getTurn() == Game.PlayerColor:
808             for n in Dice.getRollList():
809                 if piece.getSpikeNo() + n == 25:
810                     pygame.draw.rect(screen,validGreen,(0,size[1]/2,25,size[1]/2-25),0)
811                 else:
812                     highest = getHighestSpike(Game.getTurn())
813                     if piece == highest and n + piece.getSpikeNo() > 25:
814                         pygame.draw.rect(screen,validGreen,(0,size[1]/2,25,size[1]/2-25),0)
815         else:
816             for n in Dice.getRollList():
817                 if piece.getSpikeNo() - n == 0:
818                     pygame.draw.rect(screen,validGreen,(0,25,25,size[1]/2),0)
819                 else:
```

```
820                     highest = getHighestSpike(Game.getTurn())
821                     if piece == highest and piece.getSpikeNo() - n < 0:
822                         pygame.draw.rect(screen,validGreen,(0,25,25,size[1]/2),0)
823         valid_dest.pop(-1) #remove takeoff from valid_dests
824 
825     for i in valid_dest:
826         spike = getSpikeFromNumber(i) #each spike in valid_dests
827         pygame.draw.polygon(screen,validGreen,((spike.getLeftCoord()[0],spike.
        getLeftCoord()[1]),(spike.getRightCoord()[0],spike.getRightCoord()[1]),(spike.
        getCenterCoord()[0],spike.getCenterCoord()[1])),0)
828         #draw spike as green
829 
830 def checkValidPiece(piece): #check if selected piece is valid
831     if piece != None:
832         if Bar.getNoPieces(Game.getTurn()) > 0 and piece != "bar":
833             #false if don't select bar and have pieces on bar
834             tkinterMessageBox(1,"Invalid Piece Selection","Must move piece off bar first
        .")
835             return False
836         elif piece == "bar": #if you select bar
837             if Bar.getNoPieces(Game.getTurn()) == 0:
838                 #return false if there are no pieces on it
839                 return False
840             else:
841                 return True
842         elif piece == "takeoff":
843             #return false if you select the bear off bar
844             return False
845         elif piece.getContainsColor() == Game.getTurn():
846             #if spike contains your pieces
847             if len(getValidDests(piece)) > 0:
848                 #if you can make at least 1 valid move from the spike return true
849                 return True
850         elif piece.getContainsColor() != None: #if select opponents piece show error
851             tkinterMessageBox(1,"Invalid Piece Selection","Cannot select opponents piece
        .")
852         else:
853             return False
854     return False
855 
856 def checkValidDest(piece,dest,valid_dest): #check if selected destination is valid
857     if dest != None:
858         if dest != "bar": #can't select bar as destination
859             if dest == "takeoff":
860                 if dest in valid_dest: #if destination is in valid destinations
861                     return True
862                 else: return False
863             elif dest.getSpikeNo() in valid_dest: #return true if spike is in
        valid_dests
864                 return True
865     return False
866 
867 def getRemovablePiece():
868     #get a piece that can be removed from the board if possible (for AI)
869     i = 0
870     k = 0
871     while i < 2: #iterate through the board
872         while k < 12:
873             piece = Board[i][k] #try a piece
874             n = len(Dice.getRollList())
875             count = 0
876             if getClearPieces(Game.getTurn()): #if all pieces are in home quarter
877                 while count < n:
878                     if Game.getTurn() == Game.PlayerColor:
879                         if piece.getNumberOfPieces() > 0 and (piece.getSpikeNo() + Dice.
        getRollList()[count]) == 25:
880                             #if you can remove the piece directly - have pieces on spike
         6 and roll a 6
881                             return piece
882                         else:
883                             highest = getHighestSpike(Game.getTurn())
884                             if highest.getSpikeNo()+Dice.getRollList()[count] > 25 and
        piece == highest:
885                                 #have pieces on 5, not on 6. Roll a 6. Clear from 5
886                                 return piece
887                 #same for AI
888                 elif Game.getTurn() == Game.AIColor:
```

```
889                         if piece.getNumberOfPieces() > 0 and (piece.getSpikeNo() - Dice.
    getRollList()[count]) == 0:
890                             return piece
891                         else:
892                             highest = getHighestSpike(Game.getTurn())
893                             if highest.getSpikeNo()-Dice.getRollList()[count] < 0 and
    piece == highest:
894                                 return piece
895                     count+=1
896             k+=1
897         i+=1
898         k = 0
899     return False
900
901 def getRandomPiece():
902     #select a random piece (easy AI)
903     optional_pieces = []
904     for i in range(2): #iterate through board
905         for k in range(12):
906             if Board[i][k].getContainsColor() == Game.getTurn() and Board[i][k].
    getNumberOfPieces()>0:
907                 optional_pieces.append(Board[i][k])
908                 #add spike to optional_pieces if contains pieces of your colour
909     selected_valid = False
910     while not selected_valid: #loop until piece is selected which has valid dests
911         piece = random.choice(optional_pieces) #pick random piece
912         selected_valid = checkValidPiece(piece) #check if it has valid moves to make
913     return piece
914
915 def getRandomDest(piece):
916     #select a random destination (east AI)
917     while True:
918         valid_dest = getValidDests(piece) #get all valid dests
919         dest = random.choice(valid_dest)  #select a random choice
920         dest = getSpikeFromNumber(dest)
921         validDestination = checkValidDest(piece,dest,valid_dest) #check it is valid
922         if not validDestination: piece = getRandomPiece() #if not valid get a different
    piece
923         else: return dest
924
925 def canAIClear(piece):
926     #determines if the AI can remove pieces from the board - using same algorithm as
    seen in getClearPiece
927     n = len(Dice.getRollList())
928     count = 0
929     if getClearPieces(Game.getTurn()):
930         while count < n:
931             if Game.getTurn() == Game.PlayerColor:
932                 if piece.getNumberOfPieces() > 0 and (piece.getSpikeNo() + Dice.
    getRollList()[count]) == 25:
933                     return True
934                 else:
935                     highest = getHighestSpike(Game.getTurn())
936                     if highest.getSpikeNo()+Dice.getRollList()[count] > 25 and piece ==
    highest:
937                         return True
938             elif Game.getTurn() == Game.AIColor:
939                 if piece.getNumberOfPieces() > 0 and (piece.getSpikeNo() - Dice.
    getRollList()[count]) == 0:
940                     return True
941                 else:
942                     highest = getHighestSpike(Game.getTurn())
943                     if highest.getSpikeNo()-Dice.getRollList()[count] < 0 and piece ==
    highest:
944                         return True
945             count+=1
946     return False
947
948 def getInitialValidation():
949     #runs at beginning of turn to check you can actually make any moves
950     n = Bar.getNoPieces(Game.getTurn()) #gets number of pieces on the bar
951     if n != 0: valid = len(getValidDests('bar')) #if there are pieces, how many valid
    moves can you make
952     else: valid = 1
953     if valid == 0: return None #return none if no moves - turn ends
954     number_of_valid_moves = getNoValidMoves() #get number of valid moves on the board
955     if number_of_valid_moves == 0:
```

```
956             return None  #return none if aren't any - turn ends
957         return True
958 #-------------------------------------------------PIECE
        MOVE-------------------------------------------------#
959
960 def movePiece(piece,target):
961         piece.setNumberOfPieces(-1)       #-1 from number of pieces spike contains
962         target.setNumberOfPieces(1)       #Add a piece to spike
963         if target.getNumberOfPieces() == 1:
964             target.setContainsColor(Game.getTurn())
965
966 def makeTakeMove(piece,target):
967         Bar.set_piece(target.getContainsColor()) #move opposite piece to the bar
968         new_color = piece.getContainsColor()     #get new colour for destination
969         target.setContainsColor(new_color)       #change the colour the destination contains
970         piece.setNumberOfPieces(-1)              #remove 1 piece from the piece
971
972 def makeBarMove(dest):
973         dest.setContainsColor(Game.getTurn())    #set destination to contain your colour
974         dest.setNumberOfPieces(1)                #add 1 piece to the destination
975         Bar.remove_piece(Game.getTurn())         #remove a piece from the bar
976
977 def makeBarTakeMove(dest):
978         color,oppositeColor = Game.getColorOpposite()
979         Bar.remove_piece(color)                  #remove players piece from the bar
980         Bar.set_piece(oppositeColor)             #set opponents piece to the bar
981         dest.setContainsColor(color)             #set destination to contain your colour
982
983 def normalMove(piece,dest,player):
984         if Game.getTurn() == Game.PlayerColor:
985             number_used = dest.getSpikeNo() - piece.getSpikeNo() #work out the number used
        to remove from dice list
986         else:
987             number_used = piece.getSpikeNo() - dest.getSpikeNo()
988
989         if player == "player": undoSave() #save the game if it is a manual player move
990         if dest.getNumberOfPieces() == 1 and dest.getContainsColor() != Game.getTurn():
991             makeTakeMove(piece,dest)
992         else:
993             movePiece(piece,dest)
994         Dice.removeNumber(number_used)  #remove the number used to move from the dice list
995         render()
996
997 def barMove(dest,player):
998         if player == "player": undoSave()
999         if dest.getNumberOfPieces() == 1 and dest.getContainsColor() != Game.getTurn():
1000                makeBarTakeMove(dest)
1001        else:
1002            makeBarMove(dest)
1003        number_used = dest.getSpikeNo()          #get number used to remove from dice list
1004        if Game.getTurn() == Game.AIColor:
1005            number_used = 25%number_used         #adjust number to 1-6 for AI
1006        Dice.removeNumber(number_used)
1007        render()
1008
1009 def clearPieceMove(piece,player):
1010        #remove pieces from the board
1011        if player == "player": undoSave()
1012        piece.setNumberOfPieces(-1)   #remove a piece from the spike
1013        number = piece.getSpikeNo()
1014        if Game.getTurn() == Game.PlayerColor: number = 25%number #get dice number used
1015        if number in Dice.getRollList():
1016            Dice.removeNumber(number) #remove dice number
1017        else:
1018            highest = max(Dice.getRollList()) #get the highest number and remove it
1019            Dice.removeNumber(highest)
1020        render()
1021 #-----------------------------------------------TURNS--------------------#
1022
1023 def hardAI():
1024        #hard AI turn
1025        if getInitialValidation() == None:
1026            return None
1027        canClear = False
1028
1029        if Bar.getNoPieces(Game.getTurn()) > 0:
1030            piece = "bar" #make piece bar if there are pieces
```

```
1031            dest = hard_ai.canSafelyMoveBarPiece(Board,Game,Dice)
1032            #find a safe destination from the bar
1033            if not dest: #if no safe moves
1034                valid_dest = getValidDests(piece) #get a random dest
1035                dest = random.choice(valid_dest)
1036                dest = getSpikeFromNumber(dest)
1037        elif getClearPieces(Game.getTurn()): #if you can clear pieces
1038            piece = getRemovablePiece()
1039            if piece != False:
1040                dest = "takeoff"   #dest is takeoff bar
1041            else:
1042                piece = getRandomPiece()  #random piece and dest
1043                dest = getRandomDest(piece)
1044
1045        else:
1046            black_pip,white_pip = getPip()
1047            if Game.getTurn() == Game.AIColor: diff = white_pip - black_pip
1048            else: diff = black_pip - white_pip
1049            if hard_ai.getInLateGame2(1,Board,Game) == True or (diff >= 20) or (
        getClearPieces(Game.getColorOpposite()[1])):
1050                #determine if in late game strategy - prioritise furthest back pieces
1051                print("late game strategy")
1052                found_valid = False
1053                rank = 1
1054                while not found_valid:
1055                    piece = hard_ai.getHighestCurrentSpike2(rank,Game.getTurn(),Board,Game)
1056                    #keep getting furthers back piece, moving closer to home quarter until
        valid move can be made
1057                    if checkValidPiece(piece) == False:
1058                        rank+=1
1059                    else:
1060                        found_valid = True
1061                dest = getRandomDest(piece)
1062            else:
1063                #find safe moves to make
1064                piece,dest = hard_ai.getSafeMove(Board,Game,Dice)
1065                if not piece and not dest:
1066                    #if no safe moves
1067                    potential_moves = []
1068                    for i in range(3):
1069                        #get 3 random moves
1070                        piece = getRandomPiece()
1071                        dest = getRandomDest(piece)
1072                        potential_moves.append([piece,dest,hard_ai.evaluateSafety(dest.
        getSpikeNo(),Board,Game)])
1073                    #get the safest out of the 3 of them
1074                    potential_moves = sorted(potential_moves, key=lambda x: x[2])
1075                    piece,dest = potential_moves[0][0],potential_moves[0][1]
1076        #make move
1077        if dest == "takeoff":
1078            clearPieceMove(piece,"ai") #clear piece move
1079        elif piece == "bar":
1080            barMove(dest,"ai") #bar move
1081        else:
1082            normalMove(piece,dest,"ai") #normal move
1083        return True
1084
1085    def easyAI():
1086        if getInitialValidation() == None:
1087            return None
1088        canClear = False
1089
1090        #piece selection
1091        if Bar.getNoPieces(Game.getTurn()) > 0:
1092            piece = "bar"
1093            selected = True
1094        else:
1095            piece = getRandomPiece()
1096
1097        #dest selection
1098        if getClearPieces(Game.getTurn()):
1099            canClear = canAIClear(piece)
1100            if canClear:
1101                dest = "takeoff"
1102        else:
1103            dest = getRandomDest(piece)
1104        #make move
```

```
1105     if dest == "takeoff":
1106         clearPieceMove(piece,"ai") #clear piece move
1107     elif piece == "bar":
1108         barMove(dest,"ai") #bar move
1109     else:
1110         normalMove(piece,dest,"ai") #normal move
1111     return True
1112 
1113 def PlayerMove():
1114     if getInitialValidation() == None:
1115         tkinterMessageBox(1,"No valid moves","No valid moves. Skipping turn.")
1116         return None
1117 
1118     selected_piece = False
1119     while not selected_piece:
1120         #select piece
1121         piece, selected_piece = getPieceDest(selected_piece)
1122         if piece == False and selected_piece == False:
1123             return "undo"
1124         selected_piece = checkValidPiece(piece) #check if valid selection
1125 
1126     if toggleHighlightMoves.getStatus():
1127         displayValidDest(piece) #show valid destinations as green
1128         displayPieces()
1129         pygame.display.update()
1130     valid_dest = getValidDests(piece)
1131 
1132     selected_dest = False
1133     while not selected_dest:
1134         dest, selected_dest = getPieceDest(selected_dest) #destination selection
1135         if dest == False and selected_dest == False:
1136             return "undo"
1137     validDestination = checkValidDest(piece,dest,valid_dest) #check valid selection
1138     if not validDestination:
1139         return False
1140 
1141     #make move
1142     if dest == "takeoff":
1143         clearPieceMove(piece,"player") #clear piece move
1144     elif piece == "bar":
1145         barMove(dest,"player") #bar move
1146     else:
1147         normalMove(piece,dest,"player") #normal move
1148     return True
1149 
1150 def GameLoop(against_ai,hard_ai):
1151     GameOver = False
1152     Game.setInGame(True)
1153     while not GameOver:
1154         Dice.displayDice() #show the dice
1155         render()
1156         if len(Dice.getRollList()) == 0:
1157             rolled = getRollButtonEvents() #show roll dice button if no dice
1158         else:
1159             rolled = True
1160         if rolled: #if you have rolled
1161             render()
1162             pygame.display.update()
1163             turns = len(Dice.getRollList())
1164             while turns > 0: #number of moves you can make if number of dice there are
1165                 if against_ai == False: #not against ai
1166                     moved = PlayerMove()
1167                 else:
1168                     if Game.getTurn() == Game.PlayerColor:
1169                         moved = PlayerMove()
1170                     else:
1171                         if toggleAIPause.getStatus():
1172                             time.sleep(AIPAUSE/2)
1173                         if  hard_ai == False: moved = easyAI() #EASY
1174                         else: moved = hardAI()
1175                         if toggleAIPause.getStatus():
1176                             time.sleep(AIPAUSE/2)
1177 
1178                 if moved == 'undo':
1179                     tempColor = Game.getTurn()
1180                     undoLoad() #undo game
1181                     if Game.getTurn() != Game.PlayerColor and Game.getTurn() != Game.
```

```
        AIColor:
1182                            Game.setTurn(tempColor)
1183                        turns+=1
1184                    elif moved: #if you made a move
1185                        turns -=1 #minus 1 from turns
1186                        Game.setIncreaseMove() #increase the number of moves you have made
1187                    elif moved == None:
1188                        turns = 0 #no turns if no moves you can make
1189                    render()
1190                    GameOver = Game.getGameOver(Board,Bar) #see if game is finished
1191                    if GameOver:
1192                        return Game.getTurn(),Game.getMoves() #return winner and number of
        moves they made
1193                Dice.setRollList([]) #set dice list to unrolled (empty)
1194                rolled = False
1195                Game.setChangeTurn() #change whose turn it is
1196            clock.tick(32)
1197 #
        ----------------------------------------------MENU----------------------------------------------
        #
1198
1199 def displaySortOptions(c1,c2,c3,c4):
1200     #display text buttons for sorting leaderboard by
1201     firstname_txt = renderText("firstname",25,c1,10,120)
1202     surname_txt = renderText("surname",25,c2,10,150)
1203     score_txt = renderText("score",25,c3,10,180)
1204     recent_txt = renderText("most recent",25,c4,10,210)
1205     return firstname_txt,surname_txt,score_txt,recent_txt
1206
1207 def displayOrderOptions(c1,c2):
1208     #display text buttons for ordering leaderboard by
1209     ascending_txt = renderText("Ascending",25,c1,10,275)
1210     descending_txt = renderText("Descending",25,c2,10,295)
1211     return ascending_txt,descending_txt
1212
1213 def leaderBoardEvents(sortType,orderType):
1214     back_txt = renderText("Back",25,white,0,0) #back button
1215     renderText("Sort by:",25,white,0,80)
1216     renderText("Order by",25,white,0,250)
1217     if sortType == 1:
1218         firstname_txt,surname_txt,score_txt,recent_txt = displaySortOptions(validGreen,
        white,white,white)
1219     elif sortType == 2:
1220         firstname_txt,surname_txt,score_txt,recent_txt = displaySortOptions(white,
        validGreen,white,white)
1221     elif sortType == 3:
1222         firstname_txt,surname_txt,score_txt,recent_txt = displaySortOptions(white,white,
        validGreen,white)
1223     elif sortType == 4:
1224         firstname_txt,surname_txt,score_txt,recent_txt = displaySortOptions(white,white,
        white,validGreen)
1225     if orderType == 1:
1226         ascending_txt,descending_txt = displayOrderOptions(validGreen,brightRed)
1227     else:
1228         ascending_txt,descending_txt = displayOrderOptions(brightRed,validGreen)
1229
1230     while True:
1231         for event in pygame.event.get():
1232             if event.type == pygame.QUIT:
1233                 pygame.quit()
1234                 exit()
1235             mx, my = pygame.mouse.get_pos()
1236             if event.type == pygame.MOUSEBUTTONDOWN:
1237                 #handle mouse events clicking on text buttons
1238                 if back_txt.collidepoint(mx,my): return False
1239                 elif firstname_txt.collidepoint(mx,my): return 1
1240                 elif surname_txt.collidepoint(mx,my): return 2
1241                 elif score_txt.collidepoint(mx,my): return 3
1242                 elif recent_txt.collidepoint(mx,my): return 4
1243                 elif ascending_txt.collidepoint(mx,my): return 5
1244                 elif descending_txt.collidepoint(mx,my): return 6
1245         pygame.display.update()
1246
1247
1248 def addBackdrop(width,height,x,y,alpha_level):
1249     #add translucent backdrop to help objects stand-out
1250     backdrop = pygame.Surface((width,height))
```

```
1251      backdrop.set_alpha(alpha_level)
1252      backdrop.fill(grey)
1253      screen.blit(backdrop,(x,y))
1254
1255  def leaderBoards():
1256      leaderboard.loadLeaderboard()
1257      stagger_distance = 40
1258      sortType = 1
1259      orderType = 1
1260      text_color = white
1261      while True:
1262          stagger = 0
1263          screen.blit(leaderboardBackground,(0,0))
1264          addBackdrop(330,400,250,75,75) #transluscent backdrop for scores
1265          addBackdrop(200,300,0,80,75)   #transluscent backdrop for sort by
1266          sorted_lb = leaderboard.getSortedList(sortType)
1267          if orderType == 1: a,b,c,count,change_count = 0,10,1,1,1 #pointers for
        leaderboard items
1268          elif orderType == 2: a,b,c,count,change_count = len(sorted_lb)-1,len(sorted_lb)
        -11,-1,len(sorted_lb),-1
1269          if orderType == 1 and sortType == 3: a,b,c,count,change_count = 0,10,1,len(
        sorted_lb),-1
1270          if orderType == 2 and sortType == 3: a,b,c,count,change_count = len(sorted_lb)
        -1,len(sorted_lb)-11,-1,1,1
1271          for i in range(a,b,c):
1272              if orderType == 2 and sortType == 3:
1273                  if count == 1: text_color = (255, 224, 51)
1274                  elif count == 2: text_color = (166,166,166)
1275                  elif count == 3: text_color = (204, 41, 0)
1276                  else: text_color = white
1277              renderText((str(count)+") "+" ".join(sorted_lb[i])),25,text_color,size
        [0]-450,75+stagger)
1278              #show the firstname, lastname, score
1279              stagger+=stagger_distance
1280              count+=change_count
1281
1282          pygame.display.update()
1283          clock.tick(16)
1284          new_sort_type = leaderBoardEvents(sortType,orderType)
1285          if new_sort_type == False: break
1286          elif new_sort_type == 5:
1287              orderType = 1
1288          elif new_sort_type == 6:
1289              orderType = 2
1290          else: sortType = new_sort_type
1291
1292  def showInstructions():
1293      #show instructions over the screen
1294      x,y = 100,100
1295      addBackdrop(525,270,x,y,200)
1296
1297      instructions_text = open('instructions.txt','r').readlines()
1298      #open and read instructions text file
1299      stagger = 0
1300      for line in instructions_text:
1301          #for each line, render it
1302          renderText(line.strip(),15,black,x+10,y+10+stagger)
1303          stagger+=30
1304      pygame.display.update()
1305      while True:
1306          for event in pygame.event.get():
1307              if event.type == pygame.QUIT:
1308                  pygame.quit()
1309                  exit()
1310              if event.type == pygame.MOUSEBUTTONDOWN:
1311                  return
1312
1313  def settingsEvents():
1314      while True:
1315          back_txt = DisplaySettings()
1316          for event in pygame.event.get():
1317              if event.type == pygame.QUIT:
1318                  pygame.quit()
1319                  exit()
1320
1321              mx, my = pygame.mouse.get_pos()
1322              if event.type == pygame.MOUSEBUTTONDOWN:
```

```
1323                 if toggleHighlightMoves.getWithin(mx,my):
1324                     #if mouse coordinates is clicked within switch image
1325                     toggleHighlightMoves.setSwitchStatus()
1326                     #switch status - on/off
1327                 elif toggleAIDifficulty.getWithin(mx,my):
1328                     if Game.getInGame() == False:
1329                         toggleAIDifficulty.setSwitchStatus()
1330                     else:
1331                         tkinterMessageBox(1,"Error","Cannot change this setting during a
        game.")
1332                 elif togglePieceColor.getWithin(mx,my):
1333                     togglePieceColor.setSwitchStatus()
1334                 elif toggleDiceAnimation.getWithin(mx,my):
1335                     toggleDiceAnimation.setSwitchStatus()
1336                 elif toggle12Player.getWithin(mx,my):
1337                     if Game.getInGame() == False:
1338                         toggle12Player.setSwitchStatus()
1339                     else:
1340                         tkinterMessageBox(1,"Error","Cannot change this setting during a
        game.")
1341                 elif back_txt.collidepoint(mx,my): return
1342                 elif toggleAIPause.getWithin(mx,my):
1343                     toggleAIPause.setSwitchStatus()
1344
1345 def DisplaySettings():
1346     #display settings - show all the switches at their coordinates
1347     settings_text_color = white
1348     screen.blit(settingsBackground,(0,0))
1349     back_txt = renderText("Back",25,white,0,0)
1350     renderText(toggleHighlightMoves.getText(),20,settings_text_color,
        toggleHighlightMoves.getX(),toggleHighlightMoves.getY())
1351     screen.blit(toggleHighlightMoves.getImage(),(toggleHighlightMoves.getX(),
        toggleHighlightMoves.getY()))
1352     renderText(toggleAIDifficulty.getText(),20,settings_text_color,toggleAIDifficulty.
        getX(),toggleAIDifficulty.getY())
1353     screen.blit(toggleAIDifficulty.getImage(),(toggleAIDifficulty.getX(),
        toggleAIDifficulty.getY()))
1354     renderText(togglePieceColor.getText(),20,settings_text_color,togglePieceColor.getX()
        ,togglePieceColor.getY())
1355     screen.blit(togglePieceColor.getImage(),(togglePieceColor.getX(),togglePieceColor.
        getY()))
1356     renderText(toggleDiceAnimation.getText(),20,settings_text_color,toggleDiceAnimation.
        getX(),toggleDiceAnimation.getY())
1357     screen.blit(toggleDiceAnimation.getImage(),(toggleDiceAnimation.getX(),
        toggleDiceAnimation.getY()))
1358     renderText(toggle12Player.getText(),20,settings_text_color,toggle12Player.getX(),
        toggle12Player.getY())
1359     screen.blit(toggle12Player.getImage(),(toggle12Player.getX(),toggle12Player.getY()))
1360     renderText(toggleAIPause.getText(),20,settings_text_color,toggleAIPause.getX(),
        toggleAIPause.getY())
1361     screen.blit(toggleAIPause.getImage(),(toggleAIPause.getX(),toggleAIPause.getY()))
1362     pygame.display.update()
1363     clock.tick(32)
1364     return back_txt
1365
1366 def menuEvents():
1367     DisplayMenuButtons()
1368     for event in pygame.event.get():
1369         if event.type == pygame.QUIT:
1370             pygame.quit()
1371             exit()
1372
1373         mx, my = pygame.mouse.get_pos()
1374         if new.getWithin(mx,my): new.setImage(newGameHoverImg)
1375         else: new.setImage(newGameImg)
1376         if load.getWithin(mx,my): load.setImage(loadGameHoverImg)
1377         else: load.setImage(loadGameImg)
1378         if Quit.getWithin(mx,my): Quit.setImage(quitHoverImg)
1379         else: Quit.setImage(quitImg)
1380
1381         if event.type == pygame.MOUSEBUTTONDOWN:
1382             if new.getWithin(mx,my): return 1
1383             elif load.getWithin(mx,my): return 2
1384             elif Quit.getWithin(mx,my): return 3
1385             elif leaderboardButton.getWithin(mx,my): leaderBoards()
1386             elif instructionsButtonMenu.getWithin(mx,my): showInstructions()
1387             elif settingsButtonMenu.getWithin(mx,my): settingsEvents()
```

```
1388
1389 def DisplayMenuButtons():
1390     #show buttons on the menu
1391     screen.blit(new.getImage(),(new.getX(),new.getY()))
1392     screen.blit(load.getImage(),(load.getX(),load.getY()))
1393     screen.blit(Quit.getImage(),(Quit.getX(),Quit.getY()))
1394     screen.blit(leaderboardButton.getImage(),(leaderboardButton.getX(),leaderboardButton
         .getY()))
1395     screen.blit(instructionsButtonMenu.getImage(),(instructionsButtonMenu.getX(),
         instructionsButtonMenu.getY()))
1396     screen.blit(settingsButtonMenu.getImage(),(settingsButtonMenu.getX(),
         settingsButtonMenu.getY()))
1397
1398 def menuLoop():
1399     while True:
1400         screen.blit(menu_img,(0,0))
1401         option = menuEvents()
1402         if option == 1:
1403             return 'new'
1404         elif option == 2:
1405             return 'load'
1406         elif option == 3:
1407             pygame.quit()
1408             exit()
1409         pygame.display.update()
1410
1411 def getValidName(name):
1412     #validate name input
1413     name_copy = name
1414     number_of_spaces = 0
1415     for i in name:
1416         if i == " ":
1417             number_of_spaces +=1
1418     if number_of_spaces == 1:
1419         name = name.split(" ")
1420         name = [x for x in name if x]
1421         if len(name) == 2:
1422             return True
1423         else:
1424             tkinterMessageBox(1,"Invalid Name","'{}' is an invalid name. Please insert a
           first and last name. Example: 'John Smith'".format(name_copy))
1425             return False
1426     else:
1427         tkinterMessageBox(1,"Invalid Name","'{}' is an invalid name. Please insert a
         first and last name. Example: 'John Smith'".format(name_copy))
1428         return False
1429
1430 def gameOverMenu(winner,score):
1431     winner_name = ""
1432     while True:
1433         screen.blit(gameOverBackground,(0,0))
1434         text = "{} Wins! Score {}".format(winner,score)
1435         renderText(text,30,white,220,150)
1436         renderText(winner_name,40,white,380,230)
1437         pygame.display.update()
1438         for event in pygame.event.get():
1439             if event.type == pygame.QUIT:
1440                 pygame.quit()
1441                 exit()
1442             elif event.type == pygame.KEYDOWN:
1443                 import keyboard_input
1444                 letter = keyboard_input.getLetter(event.key) #get string letter from
         keyboard event
1445                 if letter!= None and letter!= True and letter!= False:
1446                     winner_name += letter #add letter to the name
1447                 elif letter == True:
1448                     valid_name = getValidName(winner_name) #check if valid when submit
         it
1449                     if valid_name:
1450                         print("Valid")
1451                         return winner_name.split(" ")
1452                     else:
1453                         print("Invalid")
1454                         winner_name = "" #delete name if not valid
1455                 elif letter == False:
1456                     length = len(winner_name)
1457                     winner_name = winner_name[:length-1]
```

```
1458
1459 #----------------------------------------MAIN
         CODE-------------------------------------------------------#
1460 if __name__ == "__main__":
1461     leaderboard = MyLeaderboard('{}//CSV//highscores.csv'.format(current_directory))
1462     Game = MyGame(PlayerColor,AIColor)
1463     undo = MyStack() #stack class for undoing
1464     screen,clock = initPygame()
1465     while True:
1466         initialise_type = menuLoop()
1467         Board = [[None for i in range(12)]for dimension in range(2)]
1468         Bar = MyBar()
1469         Dice = MyDice()
1470         initialiseBoard(initialise_type)
1471         winner,score = GameLoop(toggle12Player.getStatus(),toggleAIDifficulty.getStatus
      ())
1472 ##          winner = Game.PlayerColor
1473 ##          score = 129
1474         winner_name = gameOverMenu(winner,score)
1475         leaderboard.appendScore(winner_name[0],winner_name[1],score)
```

3.2 Classes

Listing 3.2: classButton.py

```
1  class MyButton():
2      def __init__(self,x,y,image):
3          self.x = x  #x co-ordinate of the button
4          self.y = y  #y co-ordinate of the button
5          self.image = image  #image of the button
6      def setImage(self,image):
7          self.image = image    #set image to parameter
8      def getImage(self):     #return the button's image
9          return self.image
10     def getX(self):         #return x co-ordinate of the button
11         return self.x
12     def getY(self):         #return y co-ordinate of the button
13         return self.y
14     def getWithin(self,mx,my):
15         #determine if mouse co-ordinates (mx,my) are within the button's image
16         if mx >= self.x and mx <= self.x + self.image.get_width():
17         #if the mouses x co-ordinate is within the image
18         #compare with x co-ordinate and image width
19             if my >= self.y and my <= self.y +self.image.get_height():
20             #if the mouses y co-ordinate is within the image
21             #compare with y co-ordinate and image height
22                 return True
23         return False
```

Listing 3.3: classGame.py

```
1  class MyGame():
2      def __init__(self,pc,ac):
3          self.turn = None    #who's turn in the game
4          self.PlayerColor = pc
5          self.AIColor = ac
6          self.black_moves = 0    #Number of black moves made since game started
7          self.white_moves = 0    #Number of white moves made since game started
8          self.inGame = False     #Has the game begun
9
10     def setChangeTurn(self):    #Switch turn
11         if self.turn == self.PlayerColor:   #If turn is players
12             self.setTurn(self.AIColor)      #Set turn to AI
13         else:
14             self.setTurn(self.PlayerColor)  #Set turn to players
15
16     def setTurn(self,newTurn):  #Set who's turn specifically
17         self.turn = newTurn
18
19     def getTurn(self):          #Return who's turn it is
20         return self.turn
21
```

```
22      def getInGame(self):          #Return whether the game has begun
23          return self.inGame
24
25      def setInGame(self,in_game): #set in game
26          self.inGame = in_game
27
28      def getColorOpposite(self): #returns players colour and AI colour
29          if self.turn == self.PlayerColor:   #If turn is players
30              return self.PlayerColor,self.AIColor    #return player,ai
31          else:
32              return self.AIColor,self.PlayerColor    #return ai,player
33
34      def getGameOver(self,Board,Bar):    #Get whether the game has finished
35          remaining = 0   #Number of pieces reamining to move off the board
36          for i in range(2):  #For each spike on the board
37              for k in range(12):
38                  if Board[i][k].getContainsColor() == self.getTurn():
39                      #If the spike contains pieces of your colour
40                      remaining += Board[i][k].getNumberOfPieces()
41                      #increase reamaining by the number of pieces on the spike
42          remaining += Bar.getNoPieces(self.getTurn())
43          #increase reamaining by the number of pieces you have on the bar
44          return remaining == 0
45          #return a boolean value for game over.
46
47      def setIncreaseMove(self):  #Increase the number of black moves by 1
48          if self.turn == self.PlayerColor:
49              self.black_moves+=1
50          else:
51              self.white_moves+=1
52      def getMoves(self):    #Return the number of moves the player has made
53          if self.turn == self.PlayerColor:
54              return self.black_moves
55          else:
56              return self.white_moves
57      def getBothMoves(self): #Return the number of moves both players have made
58          return [self.black_moves,self.white_moves]
59      def setBothMoves(self,b,w): #Set number of moves made for both players
60          self.black_moves = b
61          self.white_moves = w
```

Listing 3.4: classLeaderboard.py

```
1  import merge_sort,radix_sort,csv
2
3  class MyLeaderboard():
4      def __init__(self,file):
5          self.save_file = file     #csv file for reading and writing to
6          self.leaderboard = None  #un-sorted raw leaderboard list
7          self.sort_list = None    #sorted leaderboard list
8
9      def loadLeaderboard(self):  #Read the leaderboard from csv
10         scores_file = open(self.save_file)  #Open csv file
11         scores = csv.reader(scores_file,delimiter=',')  #read file
12         lb = []      #create empty leaderboard list, lb
13         for score in scores:     #for each score in the leaderboard
14             lb.append(score)     #add it to leaderboard list, lb
15         scores_file.close()
16         self.leaderboard = lb  #set leaderboard list to be lb
17
18     def appendScore(self,fn,ln,sc): #Write to the leaderboard csv
19         scores = open(self.save_file,'a',newline="") #Open the file
20         writer=csv.writer(scores)
21         writer.writerow([fn,ln,sc]) #write firstname,lastname,score
22         scores.close()
23
24     def firstname(self): #sort by firstname
25         self.sort_list = merge_sort.mergesort(self.leaderboard,0)
26     def surname(self):   #sort by surname
27         self.sort_list = merge_sort.mergesort(self.leaderboard,1)
28     def score(self):     #sort by score
29         self.sort_list = radix_sort.radix(self.leaderboard)[::-1]
30     def recent(self):    #sort by most recent
31         self.sort_list = self.leaderboard
32
33     def getSortedList(self,sort_by):
```

```
34          #Determine which way to sort the leaderboard
35          if sort_by == 1: self.firstname()
36          elif sort_by == 2: self.surname()
37          elif sort_by == 3: self.score()
38          elif sort_by == 4: self.recent()
39          return self.sort_list  #return the sorted leaderboard list
```

Listing 3.5: classToggle.py

```
1  class MyToggleSwitch():
2      def __init__(self,on,off,x_pos,y_pos,on_text,off_text):
3          self.status = True  #Boolean variable for is switch on or off
4          self.OnImage = on   #Image for when switch is on
5          self.OffImage = off #Image for when switch is off
6          self.showImage = self.OnImage   #Current image being shown
7          self.x = x_pos   #x coordinate of the image
8          self.y = y_pos   #y coordinate of the image
9          self.OnText = on_text   #Text to show when the switch is on
10         self.OffText = off_text #Text to show when the switch is off
11
12     def setSwitchStatus(self): #switch the switch on/off
13         if self.status == True: #if on
14             self.status = False #switch off
15             self.showImage = self.OffImage #set image to show as off
16         else:
17             self.status = True #if off, switch on
18             self.showImage = self.OnImage #set image to show as on
19     def getText(self):
20         #return the switch's text depending on whether switch is on/off
21         if self.status == True:
22             return self.OnText
23         return self.OffText
24     def getStatus(self):   #return switch's status (on/off)
25         return self.status
26     def getImage(self):     #return the image being shown
27         return self.showImage
28     def getX(self):     #return the x coordinate
29         return self.x
30     def getY(self):     #return the y coordinate
31         return self.y
32     def getWithin(self,mx,my):
33         #determine whether mouse coordinate is within the switch
34         if mx >= self.x and mx <= self.x + self.showImage.get_width():
35             #compare mouse x coordinate with x and image width
36             if my >= self.y and my <= self.y +self.showImage.get_height():
37                 #comapre mouse y coordinate with y and image height
38                 return True
39         return False
40
41 class MyToggleSwitchColor(MyToggleSwitch):
42     #Custom toggle switch which inherits from the main toggle switch class
43     def __init__(self,on,off,x_pos,y_pos,on_text,off_text):
44         MyToggleSwitch.__init__(self,on,off,x_pos,y_pos,on_text,off_text)
45     def getPieceColor(self):
46         #returns different colours for the pieces if on or off
47         if self.status == True:
48             return [(0,0,0),(255,255,255)]
49         return [(0,0,127),(127,0,0)]
```

Listing 3.6: classUndoStack.py

```
1  class MyStack():
2      def __init__(self):
3          self.size = 5   #maximum number of items which can be held in the stack
4          self.stack = [0 for i in range(self.size)] #stack array
5          self.front = -1     #front pointer
6          self.inStack = 0    #number of items in the stack
7
8      def push(self,item): #push items to stack
9          if self.front != self.size-1:
10             #if front pointer isn't 4
11             if self.inStack < 5: #if there is less than 5 items in stack
12                 self.inStack+=1     #add 1 to the number of items in stack
```

```
13              self.front = (self.front+1) % self.size
14              #adjust front pointer using modular arithmetics
15              self.stack[self.front] = item
16              #set item to be at the index front
17          else: #if front pointer is 4
18              self.inStack-=1 #1 less item in stack
19              self.front = -1 #set front to be -1
20              self.push(item) #add the item
21
22      def pop(self): #pop items from stack
23          empty = self.isEmpty()
24          if not empty: #if the stack isn't empty
25              self.inStack-=1 #1 less item in stack
26              self.front = (self.front-1) % self.size
27              #adjust front pointer using modular arithmetics
28              return self.stack[(self.front+1) % self.size]
29              #return the item at the previous front index
30          else:
31              return False #return false if empty
32
33      def isEmpty(self): #return boolean value for is stack empty
34          return (self.inStack == 0)
```

3.3 Other Python Files

Listing 3.7: hard_ai.py

```
1  import random
2  def getSpikeFromNumber(target_no,Board): #return spike class from a given spike number
3      i = 0
4      k = 0
5      while i < 2:
6          while k < 12:
7              if Board[i][k].getSpikeNo() == target_no:
8                  #if spikes number is the number we want
9                  return Board[i][k] #return the spike class
10             k+=1
11         i+=1
12         k=0
13
14 def getOpenPieces(Board,Game):
15     #get all the spikes which contain open pieces of your colour
16     open_pieces = []
17     for i in range(2): #for every spike on the board
18         for k in range(12):
19             if Board[i][k].getNumberOfPieces() == 1 and Board[i][k].getContainsColor()
       == Game.getTurn():
20                 #if spike has 1 piece and spike contains your colour
21                 open_pieces.append(Board[i][k]) #add spike to the open_pieces list
22     return open_pieces  #return list of open pieces
23
24 def canSafelyMoveOpenPiece(open_pieces,Board,Game,Dice):
25     #checks an open piece can be moved to a safe spike
26     if len(open_pieces) == 0: return False,False
27     safe_moves = [] #list for safe moves
28     open_pieces_length = len(open_pieces) #number of open pieces
29     roll_list_length = len(Dice.getRollList()) #number of dice left to use
30     piece_count = 0
31     dice_count = 0
32     while piece_count < open_pieces_length: #iterate through open pieces
33         while dice_count < roll_list_length: #iterate through each dice number
34             number = Dice.getRollList()[dice_count]   #get dice number to check
35             if Game.getTurn() == Game.AIColor: number*=-1
36             #If AI turn, make minus (other player moves opposite direction)
37             temp_piece = open_pieces[piece_count] #set temp_piece to be an open piece
38             temp_dest = temp_piece.getSpikeNo() + number #set temp_dest to be the spike
       the dice number away from the temp_piece
39             if temp_dest < 1 or temp_dest > 24: #if the temp_dest number isn't on the
       board (not between 1 and 24)
40                 dice_count+=1 #check the next dice number
41                 continue    #restart loop
42             temp_dest = getSpikeFromNumber(temp_dest,Board) #temp_dest is the spike
       class at index temp_dest
```

```
43              if temp_dest.getContainsColor() == Game.getTurn(): #if the temp_dest
      contains pieces of your colour
44                  safe_moves.append([temp_piece,temp_dest])   #add the piece and
      destination to the safe list
45              dice_count += 1 #check next dice number
46          piece_count+=1 #check next open piece
47          dice_count = 0 #reset dice to 0 to check from the first number again
48      if len(safe_moves) == 0:
49          return False,False
50      else: #if there are safe moves available
51          move = random.choice(safe_moves) #choose a random one
52          piece = move[0]
53          dest = move[1]
54          return piece,dest #return the piece and the destination to use
55
56  def canSafelyCoverOpenPiece(open_pieces,Board,Game,Dice):
57      #checks if a piece can be moved to cover an open piece
58      if len(open_pieces) == 0: return False,False
59      safe_moves = []
60      more_than_two = []   #list for spikes which hold more than 2 of your pieces on
61      for i in range(2): #for each spike on the board
62          for k in range(12):
63              if Board[i][k].getContainsColor() == Game.getTurn() and Board[i][k].
      getNumberOfPieces() >=3:
64                  #if spike contains your colour of pieces and contains more than 2 pieces
65                  more_than_two.append(Board[i][k])   #add spike to more_than_two list
66      more_than_two_length = len(more_than_two) #number of spikes which have more than 2
      pieces
67      roll_list_length = len(Dice.getRollList())
68      piece_count = 0
69      dice_count = 0
70      while piece_count < more_than_two_length: #iterate through spikes in more_than_two
71          while dice_count < roll_list_length: #iterate through dice rolled
72              number = Dice.getRollList()[dice_count] #get a number from dice list to
      check
73              if Game.getTurn() == Game.AIColor: number*=-1
74              #if ai turn then make minus (other player moves the other way)
75              temp_piece = more_than_two[piece_count]  #temp_piece = piece in
      more_than_two
76              temp_dest = temp_piece.getSpikeNo() + number    #temp_dest is potential
      spike the dice number away from temp_piece
77              if temp_dest < 1 or temp_dest > 24:     #if the temp_dest number isn't on
      the board (not between 1 and 24)
78                  dice_count+=1 #check the next dice number
79                  continue #restart loop
80              temp_dest = getSpikeFromNumber(temp_dest,Board) #temp_dest becomes spike
      class at board position temp_test
81              if temp_dest.getContainsColor() == Game.getTurn() and temp_dest in
      open_pieces:
82                  #if the temp_dest is in open_pieces list and contains your colour pieces
83                  safe_moves.append([temp_piece,temp_dest]) #add piece and dest to safe
      moves list
84              dice_count+=1 #check the next number in the dice list
85          piece_count+=1 #check the next piece
86          dice_count = 0 #set dice to 0 to check first number again
87      if len(safe_moves) == 0:
88          return False,False
89      else: #if there are safe moves
90          move = random.choice(safe_moves) #select a random safe move
91          piece = move[0]
92          dest = move[1]
93          return piece,dest
94
95  def canSafelyMove(Board,Game,Dice):
96      #check if a piece can be moved from one spike to another safely
97      safe_moves = []
98      pieces = []
99      for i in range(2): #iterate through board
100         for k in range(12):
101             if Board[i][k].getContainsColor() == Game.getTurn():
102                 pieces.append(Board[i][k])
103                 #get all spikes which contain pieces of your colour
104     pieces_length = len(pieces)
105     roll_list_length = len(Dice.getRollList())
106     piece_count = 0
107     dice_count = 0
108     while piece_count < pieces_length: #iterate through the spikes with your pieces
```

```
109             while dice_count < roll_list_length: #iterate through the dice numbers rolled
110                 number = Dice.getRollList()[dice_count] #get a number from dice rolled list
111                 if Game.getTurn() == Game.AIColor: number*=-1
112                 #if ai turn then make minus (other player moves the other way)
113                 temp_piece = pieces[piece_count] #temp_piece is a spike with your pieces on
114                 temp_dest = temp_piece.getSpikeNo() + number
115                 #temp_dest is a potential spike the dice number away from the temp_piece
116                 if temp_dest < 1 or temp_dest > 24: #if the temp_dest number isn't on the
        board (not between 1 and 24)
117                     dice_count+=1 #check same piece with next dice number
118                     continue #restart loop
119                 temp_dest = getSpikeFromNumber(temp_dest,Board) #temp_dest is spike class at
         board position temp_dest
120                 if temp_dest.getContainsColor() == Game.getTurn():
121                     #if temp_dest contains your colour pieces
122                     safe_moves.append([temp_piece,temp_dest]) #add temp_piece and temp_dest
        to save moves
123                 dice_count +=1 #check next dice number
124             piece_count+=1 #check next piece
125             dice_count = 0 #reset dice count to 0 to check first number again
126         if len(safe_moves) == 0:
127             return False,False
128         else: #if there are safe moves
129             move = random.choice(safe_moves) #choose a random move
130             piece = move[0]
131             dest = move[1]
132             return piece,dest
133
134 def canMoveDoubleMoveSafely(Board,Game,Dice):
135     #check if you can move a piece to a safe destination 2 moves away
136     #e.g if rolled 6 and 5, check destination 11 away
137     if len(Dice.getRollList()) != 2:
138         return False,False
139     safe_moves = []
140     i = 0
141     k = 0
142     while i < 2: #check each spike on the board
143         while k < 12:
144             if Board[i][k].getContainsColor() == Game.getTurn():
145                 piece = Board[i][k] #get piece of your colour
146                 combined_roll = Dice.getRollList()[0] + Dice.getRollList()[1]
147                 #combine the two numbers in the dice list
148                 if Game.getTurn() == Game.AIColor: combined_roll *=-1
149                 #if ai turn then make minus (other player moves the other way)
150                 double_dest = piece.getSpikeNo() + combined_roll
151                 #double_dest is destination 2 moves away from the piece
152                 if double_dest <= 24 and double_dest >= 1:
153                     #if the destination number isn on the board (between 1 and 24)
154                     double_dest_spike = getSpikeFromNumber(double_dest,Board) #
        destination becomes spike class
155                     if double_dest_spike.getContainsColor() == Game.getTurn():
156                         #if destination contains your colour pieces
157                         count = 0
158                         while count < 2: #loop to check using each number in dice list
        first
159                             number = Dice.getRollList()[count] #get dice number
160                             if Game.getTurn() == Game.AIColor: number*=-1
161                             #if ai turn then make minus (other player moves the other
        way)
162                             single_dest = getSpikeFromNumber(piece.getSpikeNo() + number
        ,Board)
163                             #single dest is destination 1 move away (dice number) from
        piece
164                             if single_dest.getContainsColor() in [Game.getTurn(),None]
        or (single_dest.getContainsColor() not in [Game.getTurn(),None] and single_dest.
        getNumberOfPieces() == 1):
165                             #if moving to single_dest is a valid move (contains your
        pieces, no pieces, or 1 of the opponents pieces)
166                                 safe_moves.append([piece,single_dest]) #add move to safe
         move
167                             count+=1 #check next dice number
168             k+=1
169         i+=1
170         k=0
171     if len(safe_moves) == 0:
172         return False,False
173     else: #if there are safe moves
```

```
174         move = random.choice(safe_moves) #choose a random safe move
175         piece = move[0]
176         dest = move[1]
177         return piece,dest
178 
179 def canMakeDoubleSeperateMoveSafely(Board,Game,Dice):
180     #check if you can make 2 seperate moves which end on the same spike
181     if len(Dice.getRollList()) != 2:
182         return False,False
183     safe_moves = []
184     i = 0
185     k = 0
186     while i < 2: #iterate the board
187         while k < 12:
188             if Board[i][k].getContainsColor() == Game.getTurn():
189                 piece = Board[i][k] #piece of your colour
190                 attempt = 0
191                 first_roll_num = 0
192                 second_roll_num = 1
193                 while attempt < 2: #will be checking twice
194                     first_roll = Dice.getRollList()[first_roll_num]
195                     #get dice number to use
196                     if Game.getTurn() == Game.AIColor: first_roll*=-1
197                     #make minus if ai (other player moves the other way)
198                     first_move = piece.getSpikeNo() + first_roll
199                     #get a potential first_move the first dice number away from the
    piece
200                     if first_move <= 24 and first_move >= 1:
201                         #if first move if on the board (between 1 and 24)
202                         first_move_spike = getSpikeFromNumber(first_move,Board) #get
    first_move spike
203                         if (first_move_spike.getContainsColor() in [Game.getTurn(),None
    ]) or (first_move_spike.getContainsColor() != Game.getTurn() and first_move_spike.
    getNumberOfPieces() == 1):
204                             #if first_move is a valid move
205                             second_roll = Dice.getRollList()[second_roll_num]
206                             if Game.getTurn() == Game.PlayerColor: second_roll *=-1
207                             #make minus if player, must check behind to see if piece can
     move to where you are now
208                             second_move = first_move + second_roll
209                             #second move is second roll away from first move in a
    negative sense
210                             if second_move <= 24 and second_move >= 1: #if second move
    is on the board
211                                 second_move =  getSpikeFromNumber(second_move,Board) #
    get second move spike
212                                 if (second_move.getNumberOfPieces() > 2 or second_move.
    getNumberOfPieces() == 1) and second_move.getContainsColor() == Game.getTurn():
213                                     #if second move has more than 2 pieces(so don't
    leave one exposed and it contains your colour pieces)
214                                     safe_moves.append([piece, getSpikeFromNumber(
    first_move,Board)])
215                                 #add piece and first_move to safe moves
216                     attempt +=1
217                     first_roll_num = 1 #switch order of dice to check
218                     second_roll_num = 0
219             k+=1
220         i+=1
221         k=0
222     if len(safe_moves) == 0:
223         return False,False
224     else: #if there are safe moves
225         move = random.choice(safe_moves) #pick a random safe move
226         piece = move[0]
227         dest = move[1]
228         return piece,dest
229 
230 def canSafelyMoveBarPiece(Board,Game,Dice):
231     #check if a piece can be moved safely off the bar
232     if Game.getTurn() == Game.PlayerColor: piece_spike = 0
233     else: piece_spike = 25
234     #get theoretical piece position on board, 0 / 25 as taken piece is off the board
235     average_dest = []
236     best_dest = []
237     for number in Dice.getRollList(): #for each number in the dice list
238         if Game.getTurn() == Game.AIColor: number*=-1
239         #make minus if AI as moves the opposite way
```

```
240             temp_dest = getSpikeFromNumber(piece_spike + number,Board)
241             #temp_dest is dest current dice number away from piece
242             if temp_dest.getContainsColor() == Game.getTurn():
243                 #if destination contains your colour pieces
244                 if temp_dest.getNumberOfPieces() > 1:
245                     #if it has more than one piece, add to average moves
246                     average_dest.append(temp_dest)
247                 elif temp_dest.getNumberOfPieces() == 1:
248                     #if it has one piece, add to best moves (as don't want to leave one open
    )
249                     best_dest.append(temp_dest)
250 
251     if len(best_dest) == 0 and len(average_dest) > 0:
252         return average_dest[0]
253     elif len(best_dest) > 0:
254         return best_dest[0]
255     else:
256         return False
257 
258 def evaluateSafety(dest,Board,Game):
259     #check safety of a move to determine how likely you are to be taken
260     distances = [6,7,8] #most dangerous distances to be away from opponent
261     if Game.getTurn() == Game.PlayerColor:
262         for i in range(len(distances)):
263             distances[i]*=-1 #make distances minus is player as checking behind
264     worst_threat = 0 #current worst move found
265     for i in range(len(distances)):
266         if dest+distances[i] > 24 or dest+distances[i] <= 0:
267             #if a destination isn't on the board, continue
268             i+=1
269             continue
270         else:
271             threat = getSpikeFromNumber(dest+distances[i],Board)
272             #get threat spike
273             if threat.getContainsColor() != Game.getTurn() and (threat.getNumberOfPieces
    () == 1 or threat.getNumberOfPieces() > 2):
274                 #if threat contains 1 piece or more than 2 pieces not of your colour
275                 if abs(distances[i]) == 7: #if the distance away from threat is 7
276                     worst_threat = 3 #worst_threat is 3 (worst)
277                 elif abs(distances[i]) == 6 or abs(distances[i]) == 8:
278                     #if distance is 6 or 8 then threat is 2
279                     if worst_threat < 2: worst_threat = 2
280             else:
281                 #otherwise worse threat is 1
282                 if worst_threat < 1: worst_threat = 1
283             i+=1
284     return worst_threat
285 #--------------------------GAME STRATEGIES--------------------------#
286 def getHighestCurrentSpike2(rank,color,Board,Game):
287     #get the spike furthers back containing your pieces
288     order_of_spikes = []
289     if color == Game.AIColor:
290         start,stop,step = 1,25,1
291     else:
292         start,stop,step = 24,0,-1
293     #start,stop,step is for reversing through the board depending on who's turn it is.
294     for i in range(start,stop,step): #iterate through board in determined order
295         spike = getSpikeFromNumber(i,Board) #get spike
296         if spike.getContainsColor() == color: #if spike contains your colour pieces
297             order_of_spikes.append(spike) #add to spike list
298     return order_of_spikes[-rank] #return the last spike in the list.
299 
300 def getInLateGame2(rank,Board,Game):
301     #determine if AI should play late game strategy
302     black_highest = getHighestCurrentSpike2(rank,Game.PlayerColor,Board,Game).getSpikeNo
    ()
303     #get the furthers back spike containing black pieces
304     white_highest = getHighestCurrentSpike2(rank,Game.AIColor,Board,Game).getSpikeNo()
305     #get the furthers back spike containing white pieces
306     return white_highest < black_highest
307     #return boolean value for if white is less than black
308 
309 #--------------------------MOVE-----------------#
310 def getSafeMove(Board,Game,Dice):
311     #order of steps to find a safe move to make
312     open_pieces = getOpenPieces(Board,Game) #get all open pieces of your colour
313     piece,dest = canSafelyMoveOpenPiece(open_pieces,Board,Game,Dice)
```

```
314     if not piece and not dest:
315         piece,dest = canSafelyCoverOpenPiece(open_pieces,Board,Game,Dice)
316         if not piece and not dest:
317             piece,dest = canMoveDoubleMoveSafely(Board,Game,Dice)
318             if not piece and not dest:
319                 piece,dest = canMakeDoubleSeperateMoveSafely(Board,Game,Dice)
320                 if not piece and not dest:
321                     piece,dest = canSafelyMove(Board,Game,Dice)
322                     if not piece and not dest:
323                         #if there are no safe moves found return false
324                         return False,False
325     return piece,dest
```

Listing 3.8: keyboard_input.py

```
1  import pygame
2  def getLetter(key):
3      #return character string based on pygame keyboard event
4      if key == pygame.K_a: return "a"  #return 'a' if a key is pressed
5      elif key == pygame.K_b: return "b"
6      elif key == pygame.K_c: return "c"
7      elif key == pygame.K_d: return "d"
8      elif key == pygame.K_e: return "e"
9      elif key == pygame.K_f: return "f"
10     elif key == pygame.K_g: return "g"
11     elif key == pygame.K_h: return "h"
12     elif key == pygame.K_i: return "i"
13     elif key == pygame.K_j: return "j"
14     elif key == pygame.K_k: return "k"
15     elif key == pygame.K_l: return "l"
16     elif key == pygame.K_m: return "m"
17     elif key == pygame.K_n: return "n"
18     elif key == pygame.K_o: return "o"
19     elif key == pygame.K_p: return "p"
20     elif key == pygame.K_q: return "q"
21     elif key == pygame.K_r: return "r"
22     elif key == pygame.K_s: return "s"
23     elif key == pygame.K_t: return "t"
24     elif key == pygame.K_u: return "u"
25     elif key == pygame.K_v: return "v"
26     elif key == pygame.K_w: return "w"
27     elif key == pygame.K_x: return "x"
28     elif key == pygame.K_y: return "y"
29     elif key == pygame.K_z: return "z"
30     elif key == pygame.K_SPACE: return " "
31     elif key == pygame.K_BACKSPACE: return False #return false if backspace key is
       pressed
32     elif key == pygame.K_RETURN: return True #return true if enter key is pressed
```

3.4 Sorting Algorithms

Listing 3.9: merge_sort.py

```
1  def merge(a,b,ob):
2      merged = [] #merged list
3      while len(a) != 0 and len(b) != 0: #while both arrays are not empty
4          if a[0][ob] < b[0][ob] or a[0][ob] == b[0][ob]:
5              #if item at index 0 in a is less than b
6              merged.append(a.pop(0))
7              #pop 0th element of a and append to merge list
8          elif a[0][ob] > b[0][ob]:
9              #if item at index 0 in b is less than a
10             merged.append(b.pop(0))
11             #pop 0th element of b and append to merge list
12     if len(a) !=0 and len(b) == 0:
13         #if a is not empty and b is
14         merged += a
15         #add rest of a to merge list
16     elif len(a) == 0 and len(b) != 0:
17         #if b is not empty and a is
18         merged += b
19         #add rest of b to merge list
```

```
20      return merged
21
22  def mergesort(array,order_by):
23      if len(array) == 0 or len(array) == 1:
24          return array #return array if length 0 or 1
25      else:
26          middle = int(len(array)/2) #find middle of the array
27          a = mergesort(array[:middle],order_by) #call this function using left of the
        array
28          b = mergesort(array[middle:],order_by) #call this function using right of the
        array
29          return merge(a,b,order_by) #merge array a and b
```

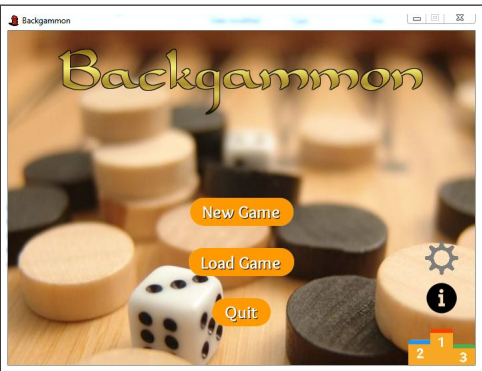
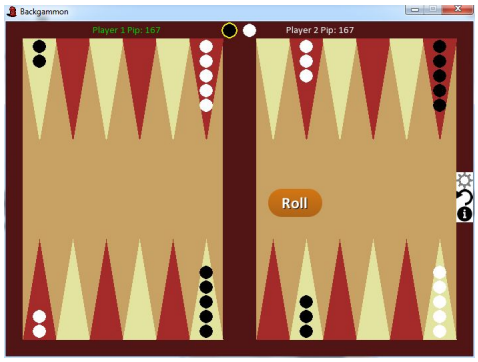
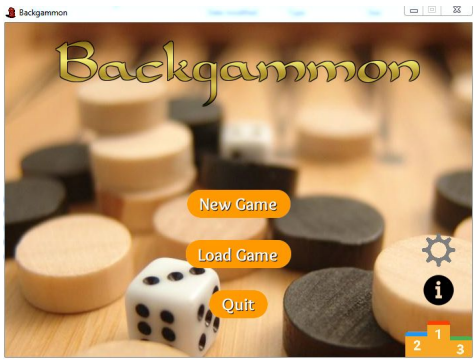
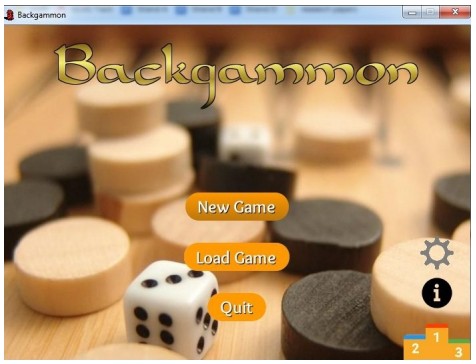
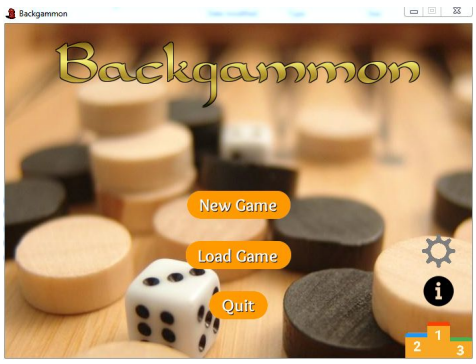
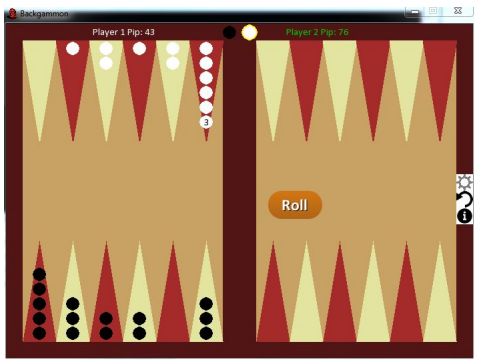
Listing 3.10: radix_sort.py

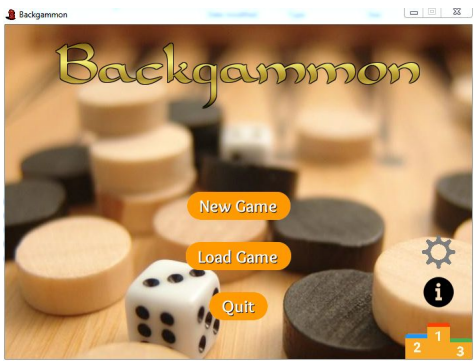
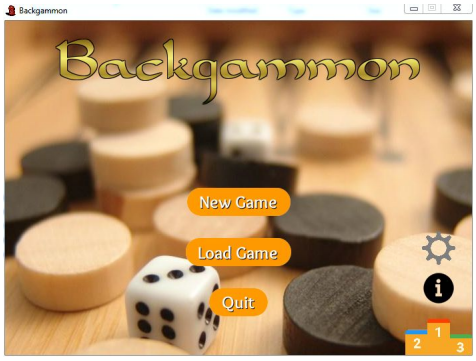
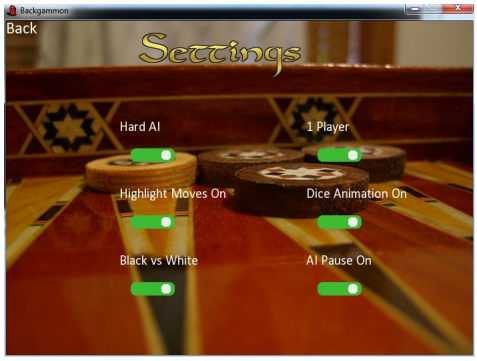
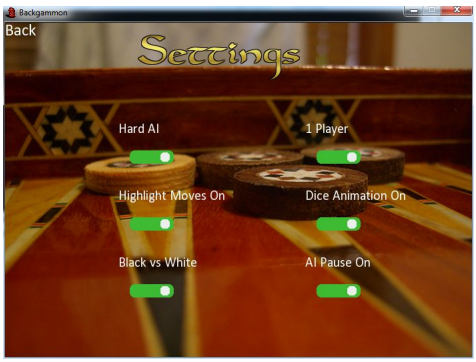
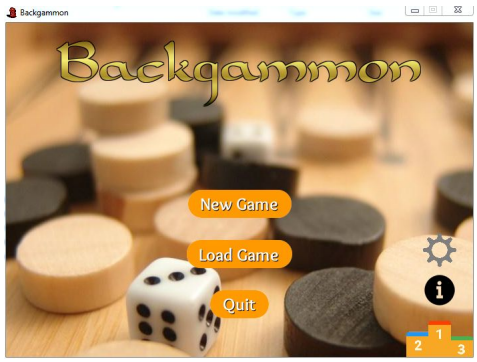
```
1   def getDig(n,modder,dig_array,maxLength,num):
2       #recursive function to get all the didgits in a number
3       #e.g - 10=[1,0], 100=[0,0,1]
4       dig = n % modder #get digit at end (10=0,345=5)
5       actual_dig = int(dig / (modder/10))
6       #divide the digit by what you modded it by divided by 10
7       modder *= 10 #times the modder by 10
8       new_n = n - dig #new number is number - digit
9       dig_array.append(actual_dig) #add actual digit to the array
10      if new_n == 0 and len(dig_array) == maxLength:
11          #return the digits if new_n = 0 and length of digits is maxlength
12          num[2] = dig_array
13          return num
14      #call this function again with the new number and new modder
15      return getDig(new_n,modder,dig_array,maxLength,num)
16
17  def fillBuckets(n,digits_array,radix_buckets):
18      for i in range(len(digits_array)): #iterate through all the digits of all the
        numbers
19          radix_buckets[digits_array[i][2][n]].append(digits_array[i])
20          #append to bucket digits[n] in radix_buckets
21          #example: number 321, digits [1,2,3]. In bucket 1 first time, 2 second time, 3
        third time
22      new_digits_array = []
23      for bucket in radix_buckets:
24          #for each bucket
25          while len(bucket) != 0:
26              new_digits_array.append(bucket.pop(0))
27              #empty bucket into new_digits_array
28      return new_digits_array
29
30  def magic(numList):
31      #reverse digits array to and convert number
32      #example: [3,2,1] = 123
33      sorted_list = []
34      for nums in numList:
35          s = ''.join(map(str, nums[2][::-1]))
36          nums[2] = s
37          sorted_list.append(nums)
38      return sorted_list
39
40  def radix(array):
41      radix_buckets = [[] for i in range(10)] #create 2d array of 10 elements
42      digits_array = []
43      ints_array = []
44      for i in array:
45          ints_array.append(i[2]) #add all the numbers to ints_array
46      maxInt = 0
47      #find the largest number in ints_array
48      for i in ints_array:
49          if int(i) > maxInt:
50              maxInt = int(i)
51      maxInt = len(str(maxInt))
52      #get number of digits in the largest number - e.g. 10=2, 100=3
53      for num in array:
54          digits_array.append(getDig(int(num[2]),10,[],maxInt,num))
55      for i in range(maxInt): #iterate number of digits in largest number
56          digits_array = fillBuckets(i,digits_array,radix_buckets)
57      return magic(digits_array)
```

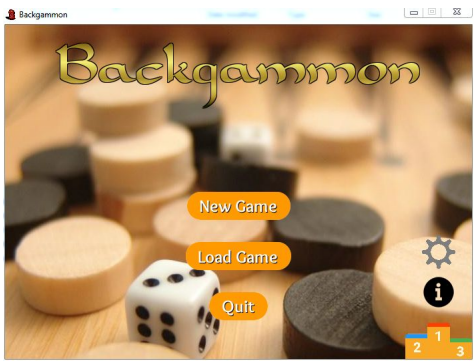
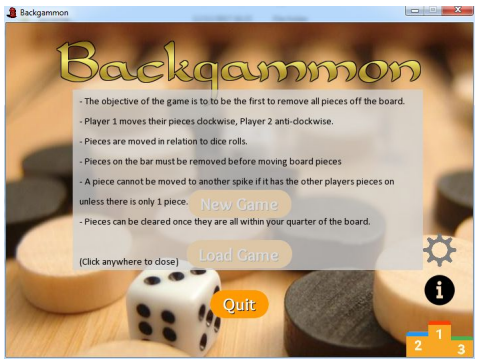
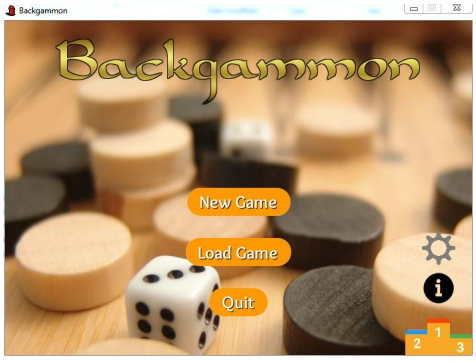
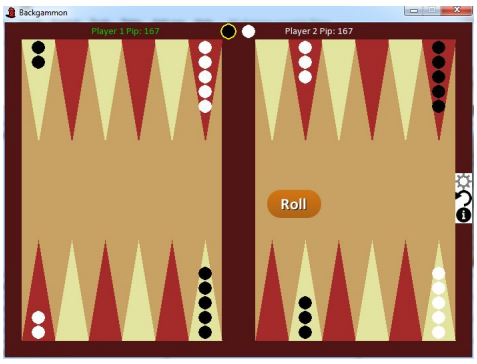
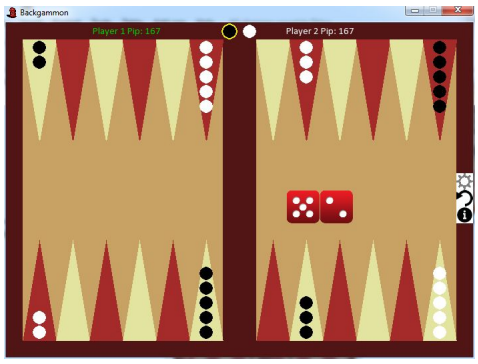
4 Testing

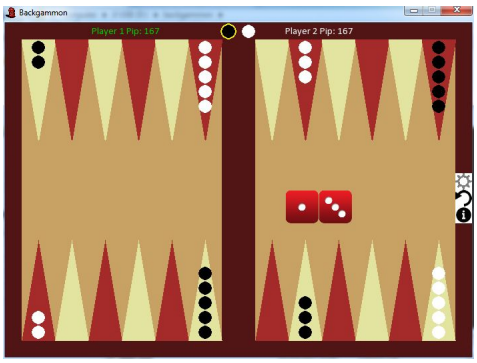
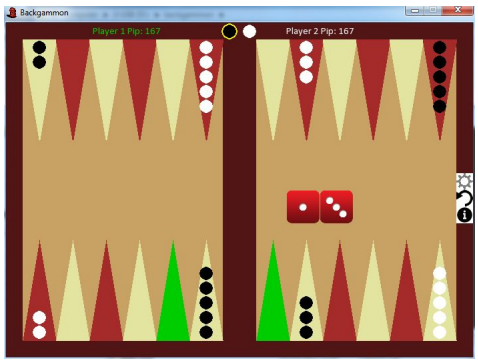
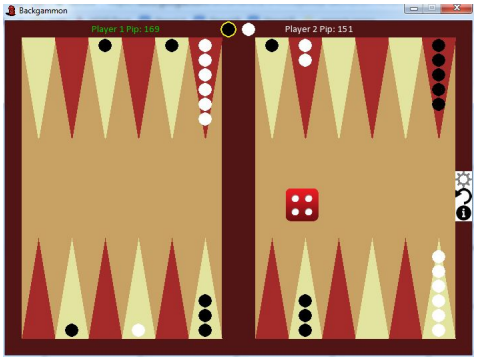
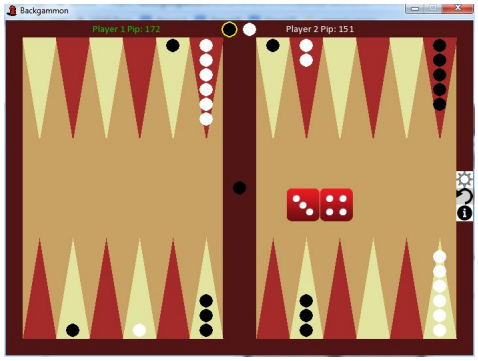
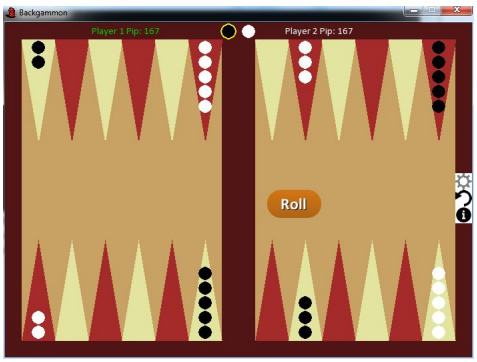
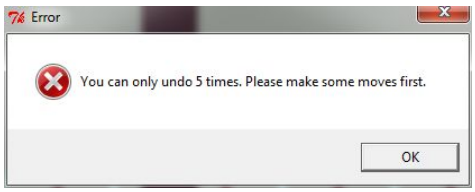
4.1 Testing Table

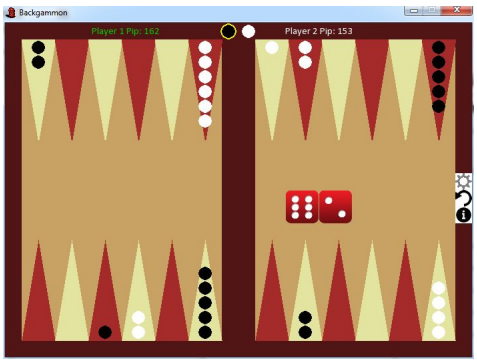
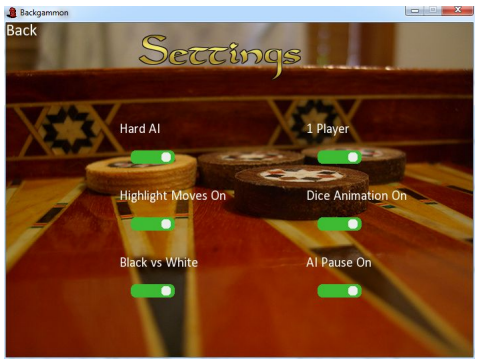
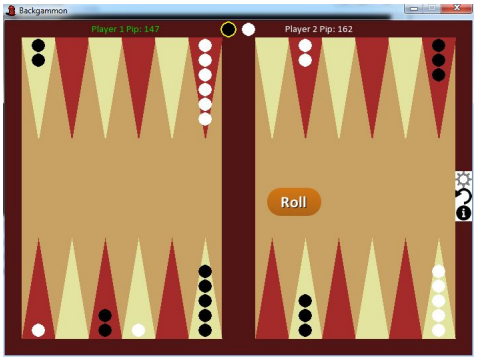
Table 4.1: Testing table with screenshots from my finished game.

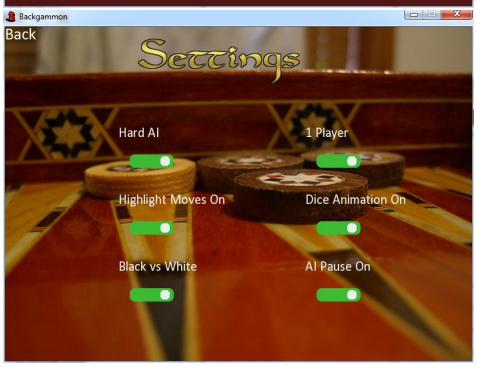
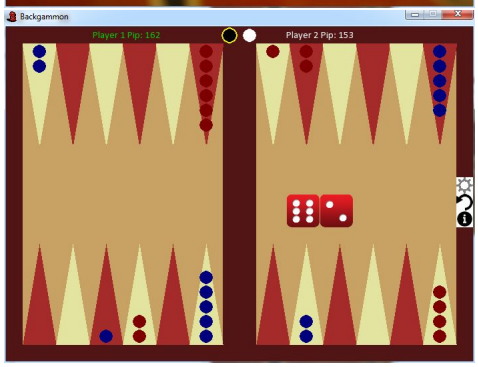
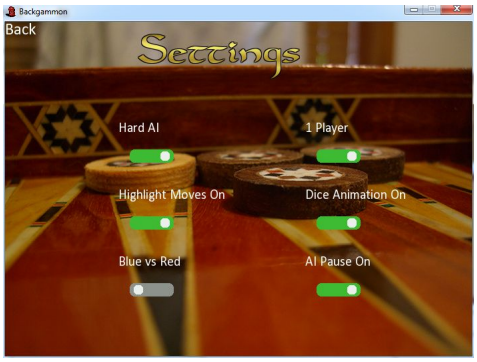
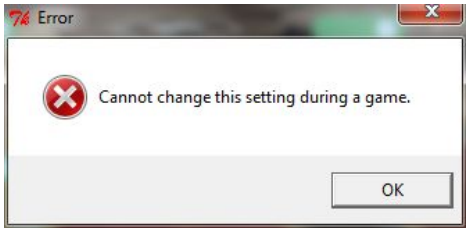
Test name	Before	After	Pass
New game button			✓
Button hover			✓
Load game button			✓

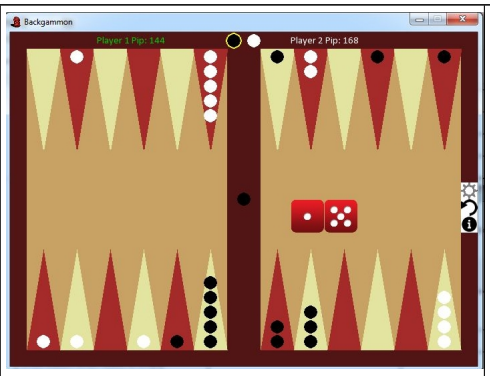
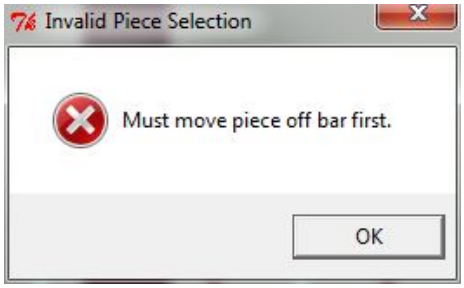
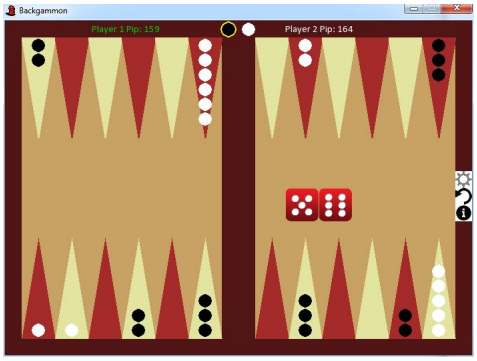
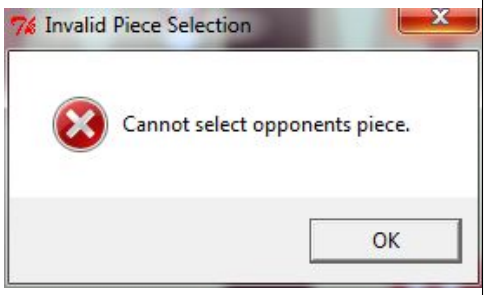
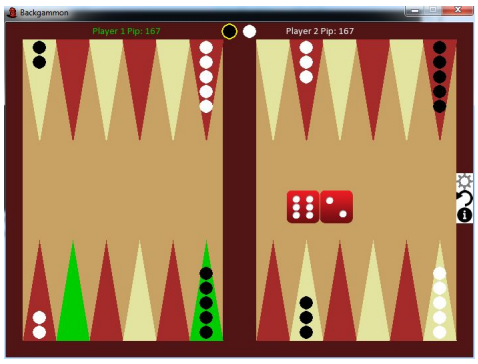
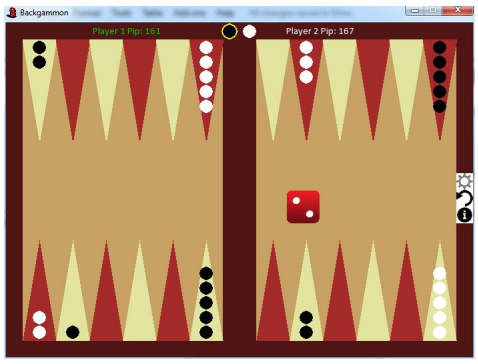
Test name	Before	After	Pass
Quit button		N/A	✓
Settings button			✓
Settings back button			✓

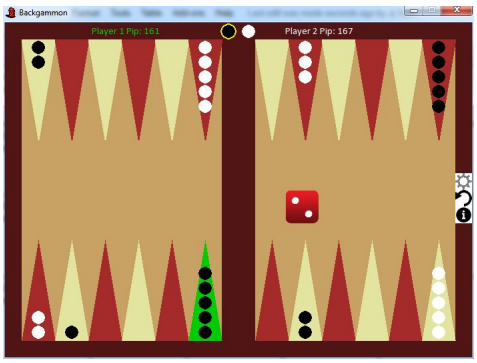
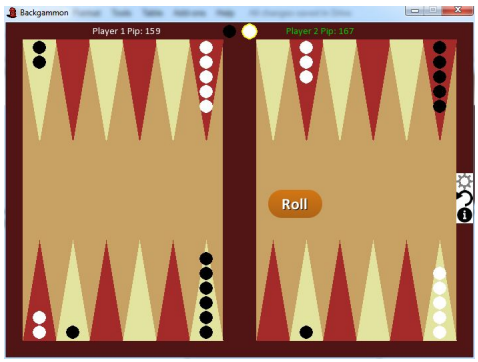
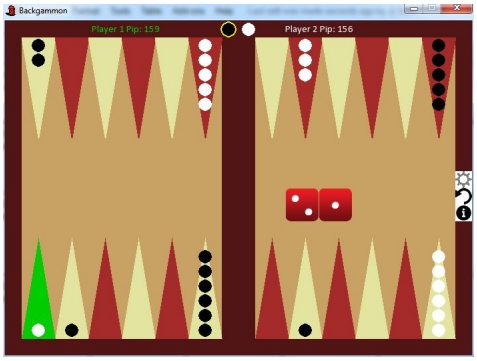
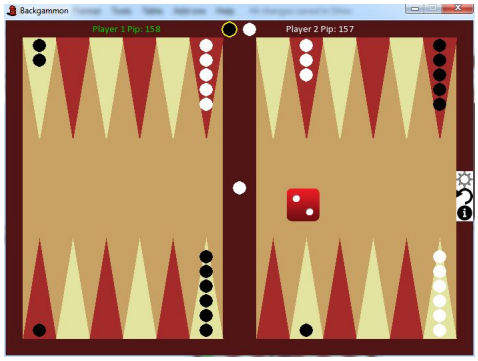
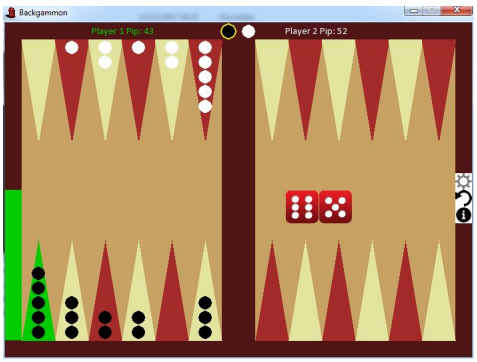
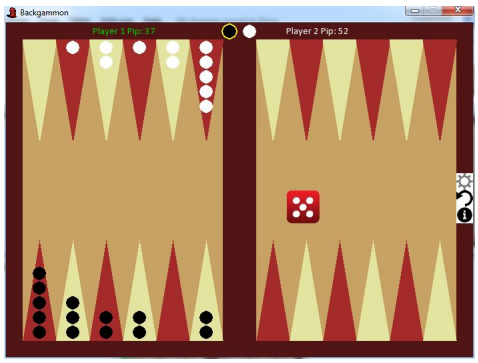
Test name	Before	After	Pass
Instructions button			✓
Leaderboards button			✓
Roll button			✓

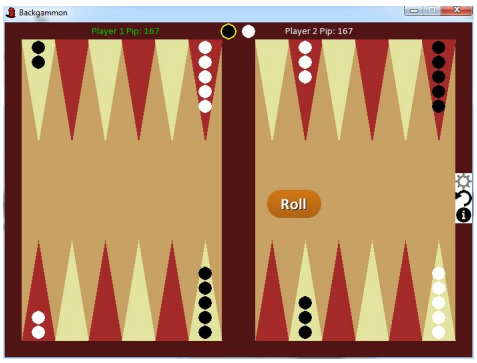
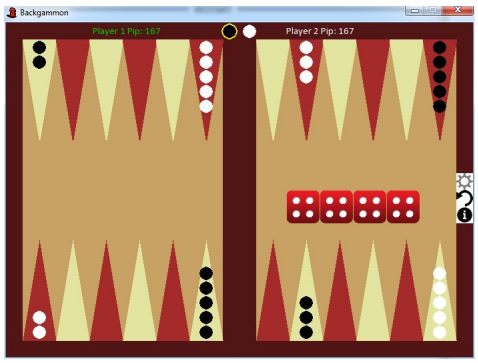
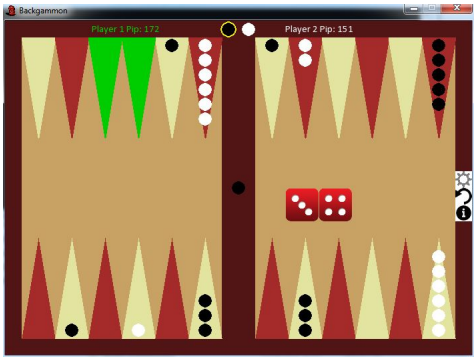
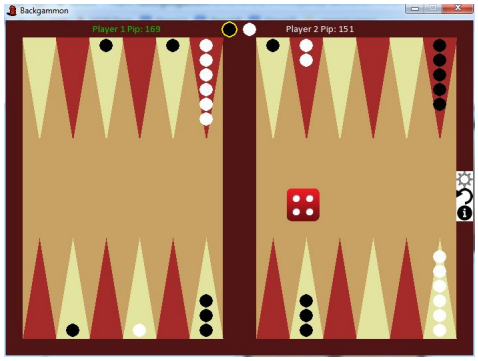
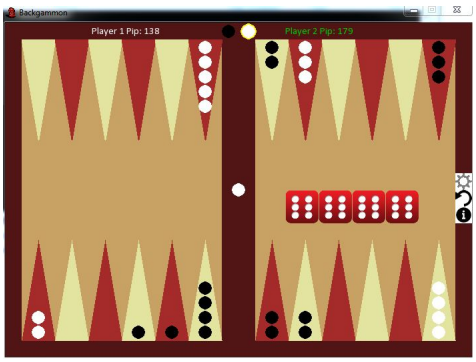
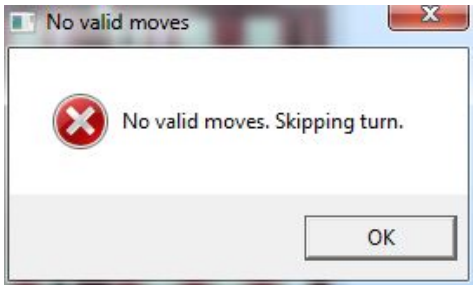
Test name	Before	After	Pass
Piece Selection			✓
Undo button			✓
Undo button when stack is empty			✓

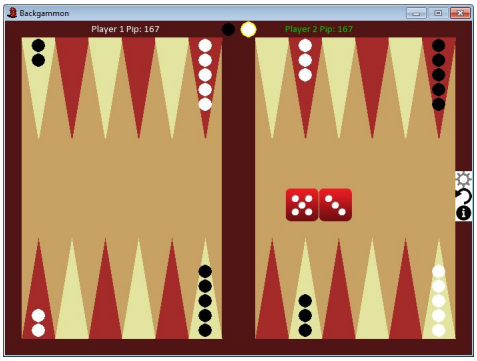
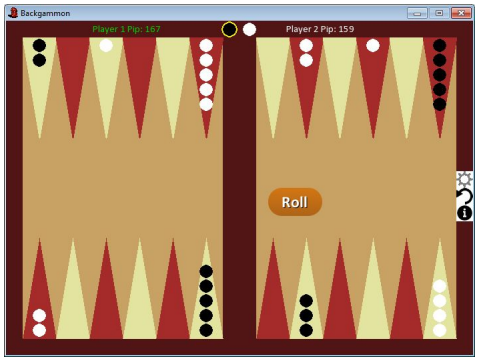
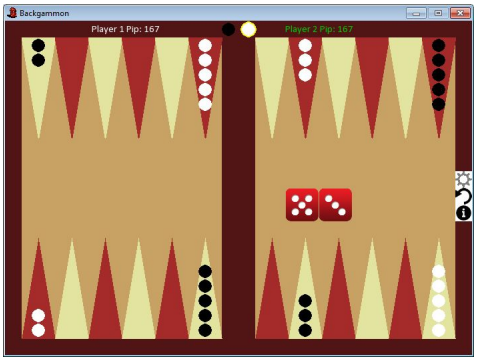
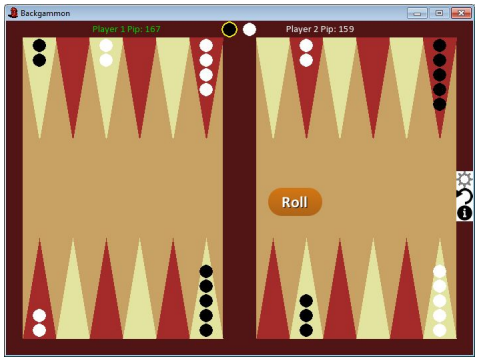
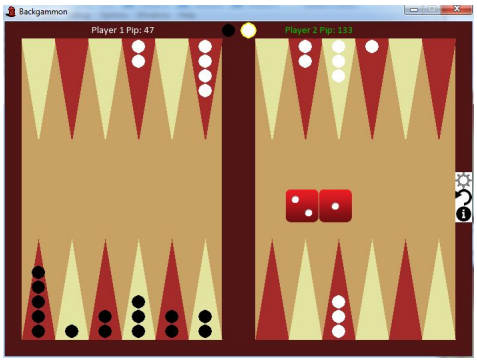
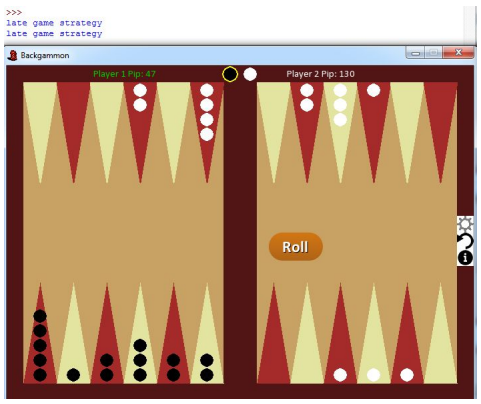
Test name	Before	After	Pass
Settings button in game			✓
Instructions button in game			✓

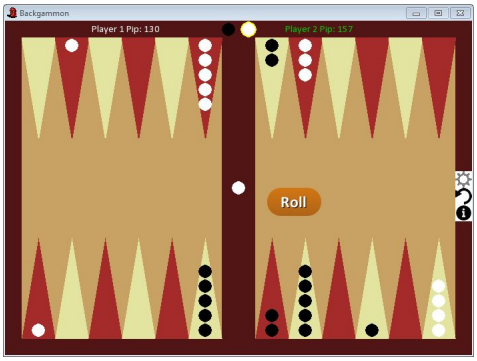
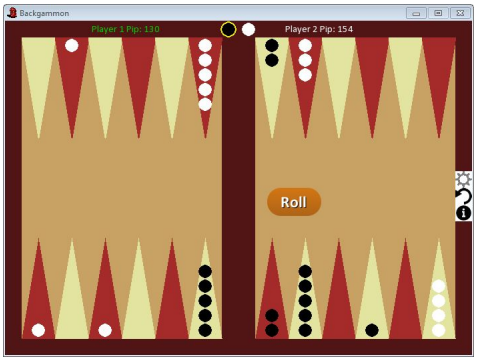
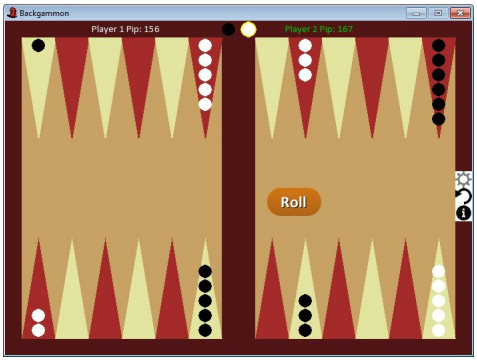
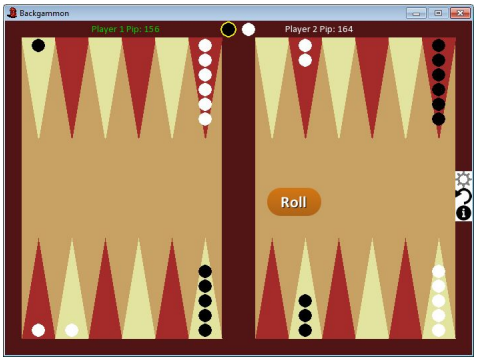
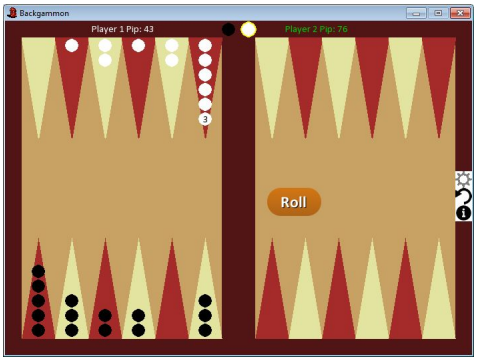
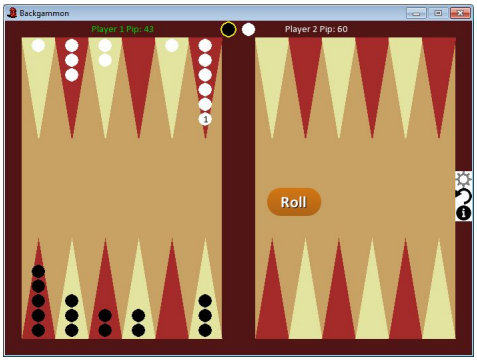
Test name	Before	After	Pass
Toggle settings piece colour in game	 	 	✓
Toggle settings AI difficulty in game			✓

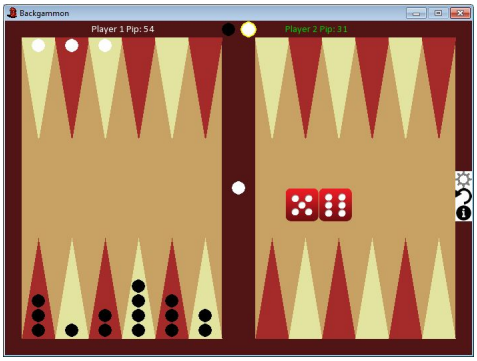
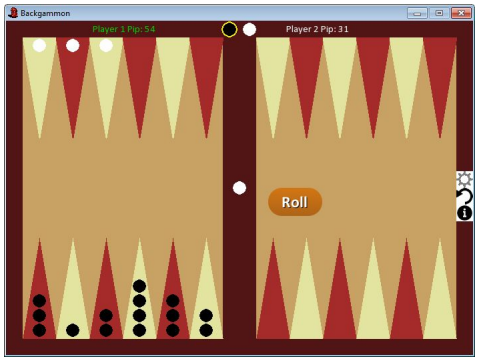
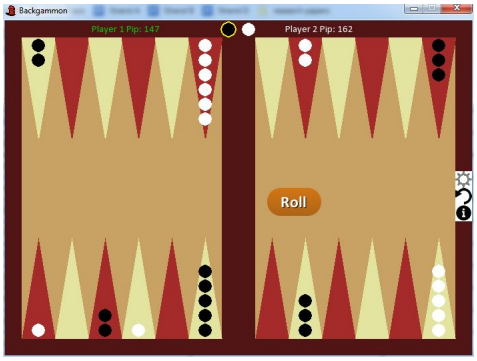
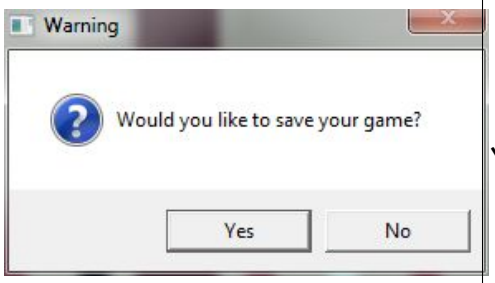
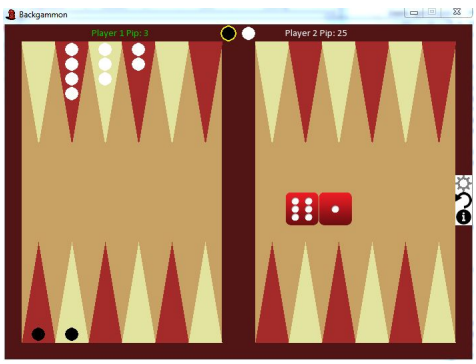
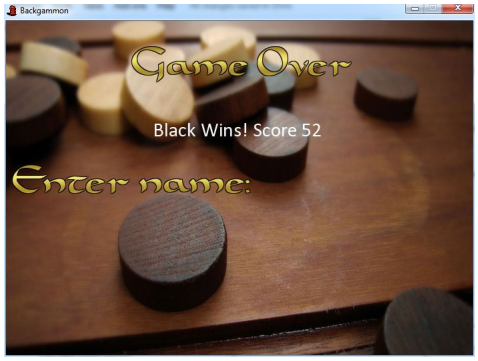
Test name	Before	After	Pass
Invalid piece selection I			✓
Invalid piece selection II			✓
Destination selection I			✓

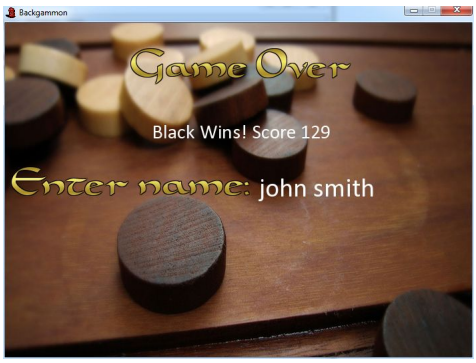
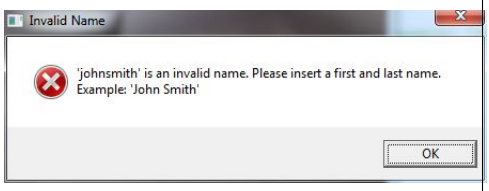
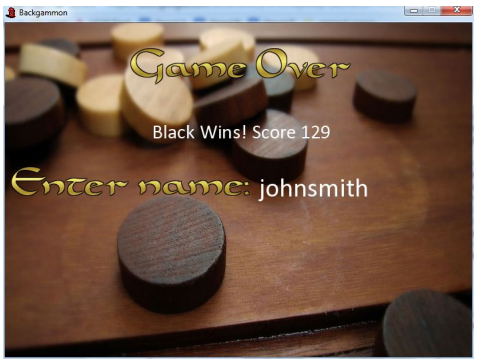
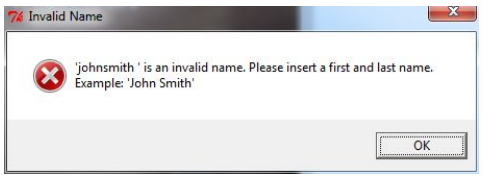
Test name	Before	After	Pass
Destination selection II			✓
Destination selection III			✓
Destination selection IV			✓

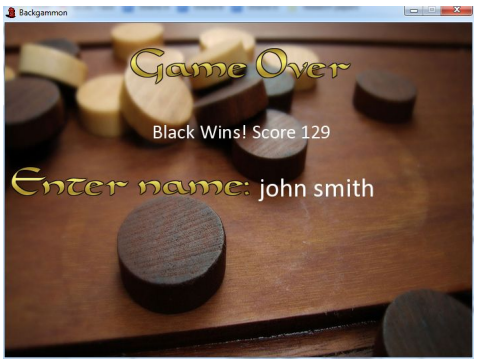
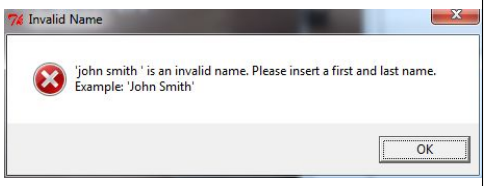
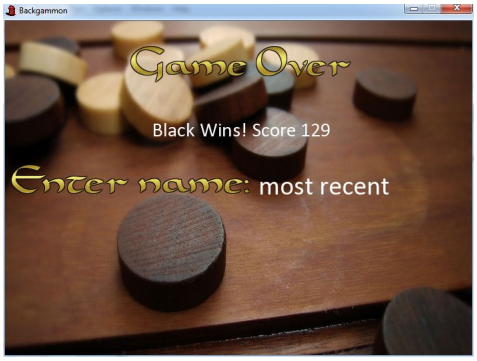
Test name	Before	After	Pass
Roll a double			✓
Taken piece			✓
No valid moves			✓

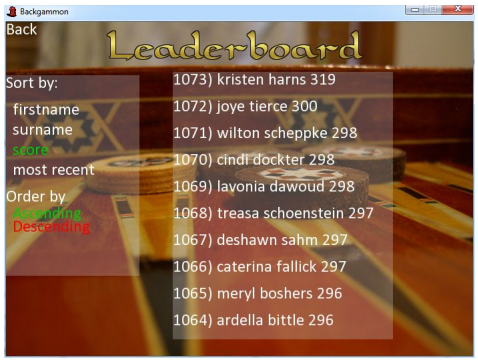
Test name	Before	After	Pass
Easy AI			✓
Hard AI			✓
Hard AI late game strategy			✓

Test name	Before	After	Pass
AI taken			✓
AI normal move			✓
AI bear off move			✓

Test name	Before	After	Pass
AI no valid moves			✓
Exit and save			✓
Game over screen			✓

Test name	Before	After	Pass
Valid name input			✓
Invalid name input I			✓
Invalid name input II			✓

Test name	Before	After	Pass
Invalid name input III			✓
Leaderboard sort recent			✓
Leaderboard sort first-name	N/A		✓

Test name	Before	After	Pass
Leaderboard sort surname	N/A		✓
Leaderboard sort score	N/A		✓
Leaderboard order ascending	N/A		✓

Test name	Before	After	Pass
Leaderboard order descending	N/A		✓

4.2 Demonstration Video

As well as screenshots, I will also make a video of my game which I will upload to Youtube. This will help make it clearer to see and understand the gameplay and the function of my AI which may not be clear from a screenshot.

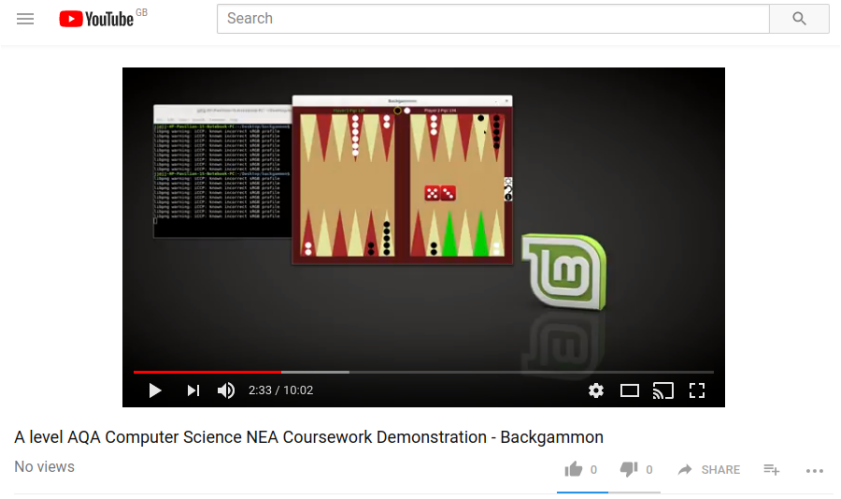


Figure 4.1: Demonstration video: `https://youtu.be/4Ec_ursmFEo`

5 Evaluation

5.1 Effectiveness of solution

My project was aimed at creating a game of backgammon that my client could use to teach his younger students old classic board games. I think my project was effective in doing this as I managed to complete my objectives which I set out - Creating a menu, creating a game of backgammon which can be played 2 player or against an AI with 2 difficulties, ensuring the AI and player had to follow the same rules and creating a leaderboard system for the students to add a competitive feel to the game.
Creating 2 different levels of AI to play against was a challenge as I had to ensure there was a significant difference in difficulty between them. To do this I had to really think about the unconscious processes and decisions that I make when I make a move. I then had to turn this process into a decision tree, creating algorithms to imitate the kind of decisions I would make.

5.2 Improvements

If I were to redo my project, there are a number of improvements I would like to make to my code.

1. **Highlight last move**
   I think highlighting the last moves made could be really useful to players who may have missed the move that the AI made.

2. **Separate leaderboards**
   It would be sensible to add separate leaderboards for games won in 2 player mode, against the easy AI and for the hard AI. This is because currently someone could appear higher for beating the easy AI than someone who beat the hard AI which I don't think is fair. In addition I think it could also be a good idea to add scrolling to the leaderboard so you could view more than just the top 10 in each category.

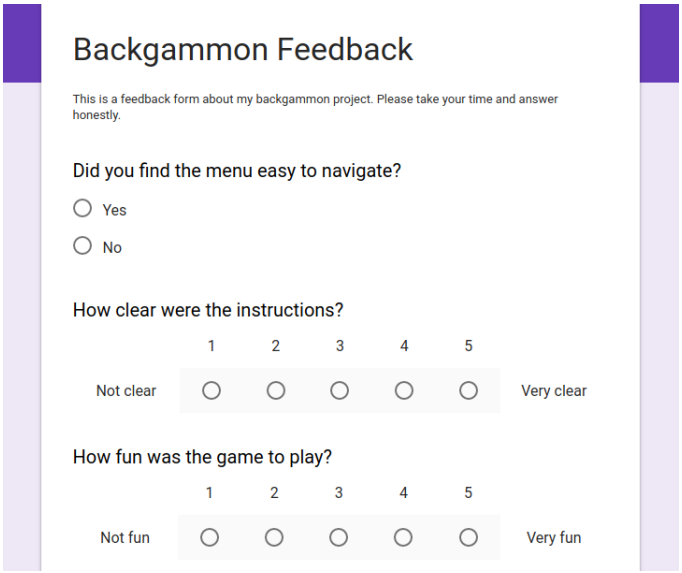
3. **Game Animations**
   I would also like to add animations to the pieces when they move so appears the are sliding across the board. While this is not an essential feature to the functioning of the game, it would add to the game experience making to more aesthetically pleasing to see while playing.

4. **Sound effects / background music**
   Sound effects such as the rolling of dice and the movement of pieces would be a nice feature to add to the game to make it more fun. Furthermore, general background music would make the game more immersive and enjoyable to play.

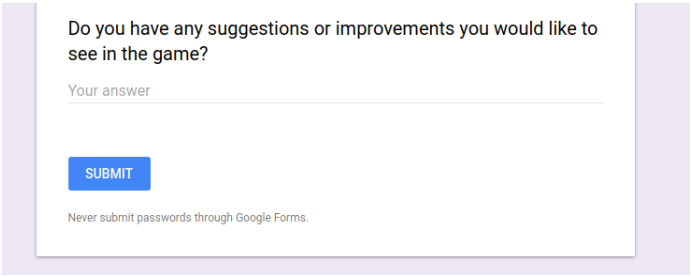
5. **In game hints**
   I would like to add a hints button, where if you are stuck about what move to make, it will highlight a suggested move based off the safe move checks my hard AI makes. I think this would be really beneficial to beginners who haven't played the game before to help them understand the different types of moves to make in different situations.

5.3 Third party feedback

5.3.1 Feedback form



(a)



(b)

Figure 5.1: Feedback questionnaire

5.3.2 Responses

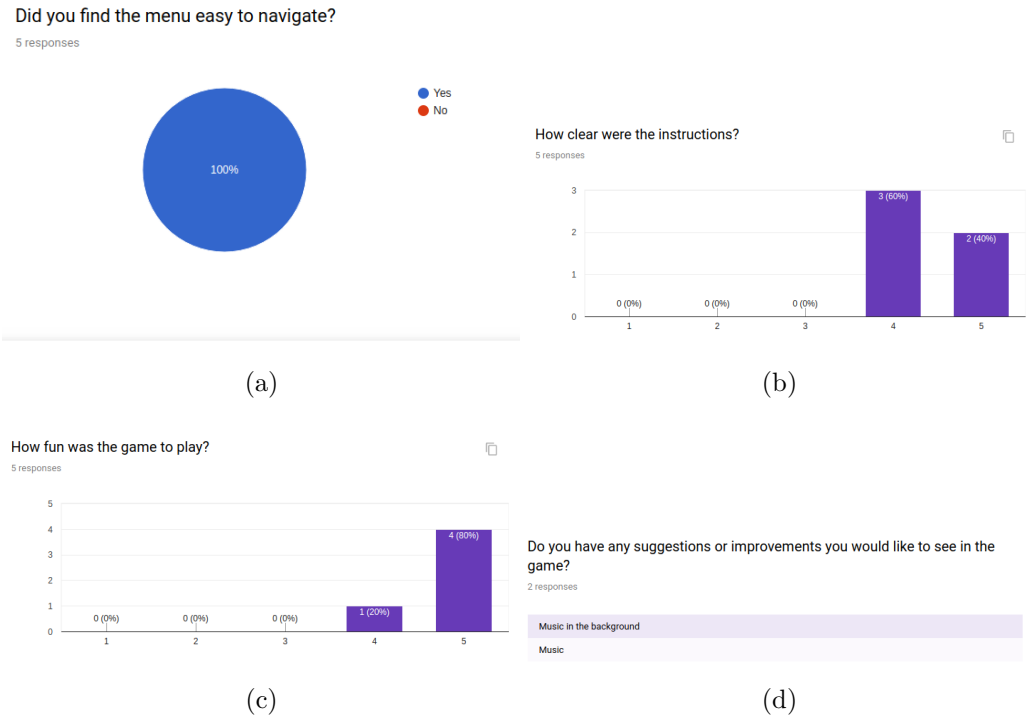


Figure 5.2: Responses from my feedback form (see figure 5.1)

5.3.3 Feedback from client

Overall I am very happy with the solution. The game functions very well and the students will all be able to play this whether they are beginners or are already familiar with the game. I am also impressed at how well the AI performs in easy and in hard mode. Finally I like the feature of the settings menu so the students can tailor the game to make the most out of their experience, enjoy it, and play more.