SOFTWARE DESIGN AND DEVELOPMENT

YEAR 11 2021

TASK 2

PORTFOLIO

SARAH CHENG

CONTENTS

DEFINING AND UNDERSTANDING THE PROBLEM

BRIEF

Defining the problem is vital to the successful development of this project.

The brief of this project is outlined on this page, which presents the problem to be solved for this project.

Optime software http://www.optimesoftware.com/ are looking for new developers to enter the team in developing interactive games for IOS and Android devices.

In particular they are looking for someone to develop their new app based on the game Snakes and Ladders. In order to apply for the position you are to design and complete your own Snakes and Ladders game for the software company.

Your Snakes and Ladders application is to follow the structured development approach to create the application.

You are required to understand a 2-Dimensional array to complete this project and as a base to have a functioning Snakes and Ladders allowing for two or more players to play against each other. You are to use at least one data structure for this project and work effectively on your documentation to complete the project.

GAME STORY

- *Each player has a different coloured token, with their order of playing determined before the game starts.*
- *Each player starts with their token outside the board.*
- *Players take turns to roll the dice*
- *The player moves their token forward the number of spaces shown on the dice*
- *If the token lands at the bottom of a ladder, it moves to the top of the ladder*
- *If the token lands on a snake's head, it moves down to the snake's tail*
- *The first person to exactly land on the final square is the winner*

Source: https://learnenglishkids.britishcouncil.org/crafts/snakes-and-ladders

AUDIENCE

Players: 2 or more

Age range: 5+

Skills required: Counting, observation

Source: Google

GUI

- *Modern, aesthetically pleasing*
- *Appropriate to players of young ages*
- *Design of GUI elements are to stay within the family-friendly genre*

INDEPENDENT RESEARCH

It is then necessary to get a thorough and precise understanding of the problem, which involves time and research. In this case, similar software solutioons and non-computer based solutions can be examined and are particularly helpful. Therefore, independent research was carried out to gain an undertstanding on the game story, audience, and GUI.

OBJECTIVES

A list of requirements or objectives should be formulated as they are essential to a complete understanding of the problem. The objectives for this project are listed on the following spread.

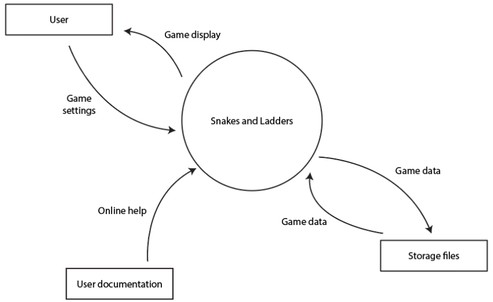
These ensure that my understanding of the problem will meet the needs of the target audience, as they can also be used for the evaluation of the final product.

Graphical User Interface and Feedback	*Control and Data Structures*	*Advanced Features*	*Error Free Solution*
The application must adhere to current trends in user interface development; must be aesthetically pleasing and family-friendly.	Appropriate control structures must be implemented into the program, ranging from sequence, selection and repetition.	The user must be able to save and resume incomplete games.	The solution should be a functional Snakes and Ladders game in considering the initial objectives and free of errors.
All navigation must be appropriate and error free.	Appropriate data structures must be implemented into the program. This should be either arrays or records.	The game must have at least 3 different board set-ups.	
The selection of appropriate interface objects and graphics must be consistent and professional in nature.	A 2-Dimensional array must be used.	The game must have a dice animation.	
The application must provide users with appropriate and relevant feedback. This must assist the user to feel in control of the application and must include aspects like alerts, notifications and labels.		The game must be capable of recording and updating the high score for each board type. This must be accessible to the user as well.	

CONTEXT DIAGRAM

Obtaining an understanding of the problem will allow for a list of required outputs to be required, which are the result of processed inputs.

The following context diagram shows the external inputs and outputs into and out of the system for this project - the Snakes and Ladders process.



IPO

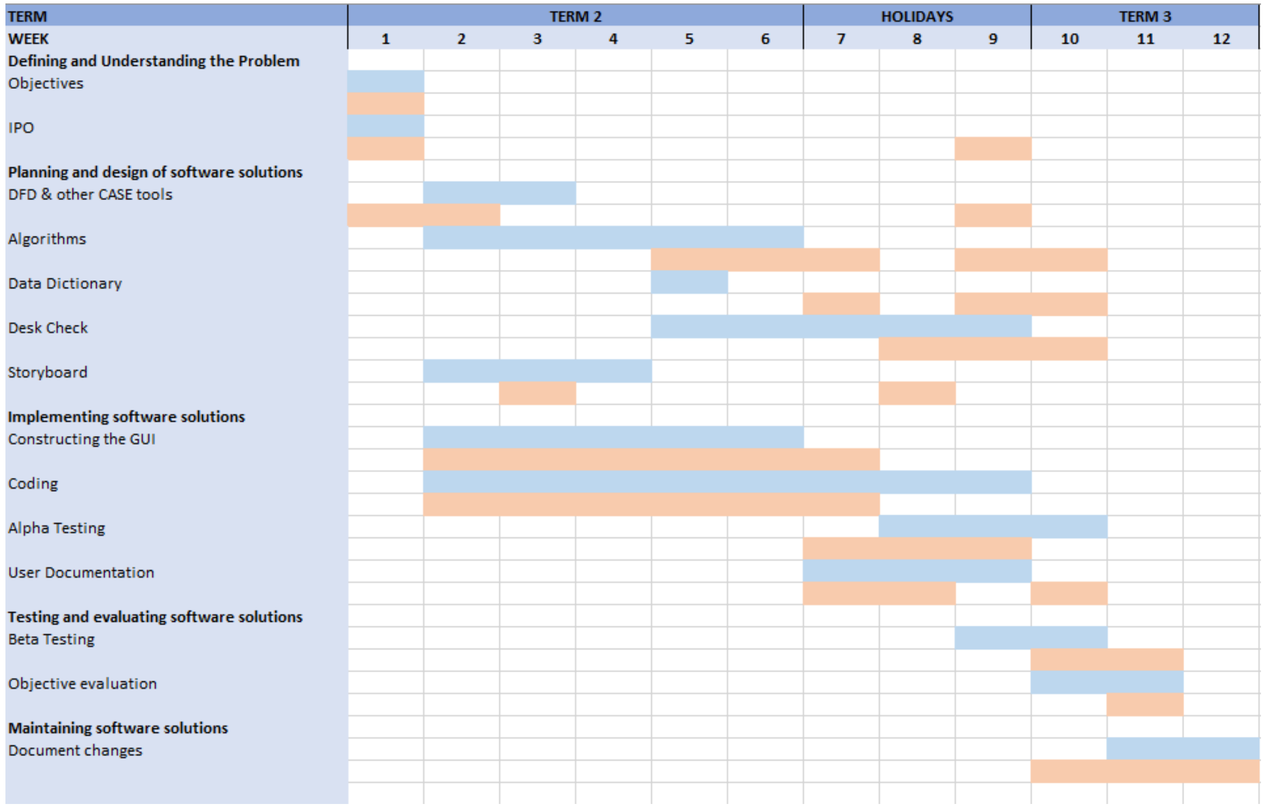
After determining the inputs and outputs, the nature of the required processes is considered. These processes transform the inputs into the outputs.

This is displayed as an IPO diagram, in the form of a table with 3 columns for Input, Process and Output. The following IPO diagram describes the Snakes and Ladders system for this project.

Input	Process	Output
	Directs user to select game settings	
Number of players	Saves the player count as an integer variable	
Player colours	Allows the user to select the colours within the limit of the player count Assign each player number to a colour in the order they were selected	
Board size	Generates board according to selected size	
Name of new game	Creates a new text file named after the user input Writes game data to a text file	A new text file with the name as the name of the game A new game - spawns the selected number of players with their colours in the order they were chosen, and the board of the selected size
Game progress from a saved-game file	Loads the data of a selected game from text file Generates the game using the input data	A past game
	Rolls the die (generates a random number between 1 and 6) Updates a player's location according to the number rolled Checks for victory Increases the player's move count by 1 Updates to have the next player in turn to move	Movement of player on the board
Highest score (from text file)	To be checked with a victory score for a new high score	
Winner's number of turns for the game	Checks score against highest score recorded Overwrites with the new high score if it's lower than the recorded score	Loads and continues a selected past game in progress
	Opens the instructions page	

GANTT CHART

A gantt chart was developed to illustrate the project schedule. This helped with planning each stage of the structued approach as well as tracking the progress of the project.

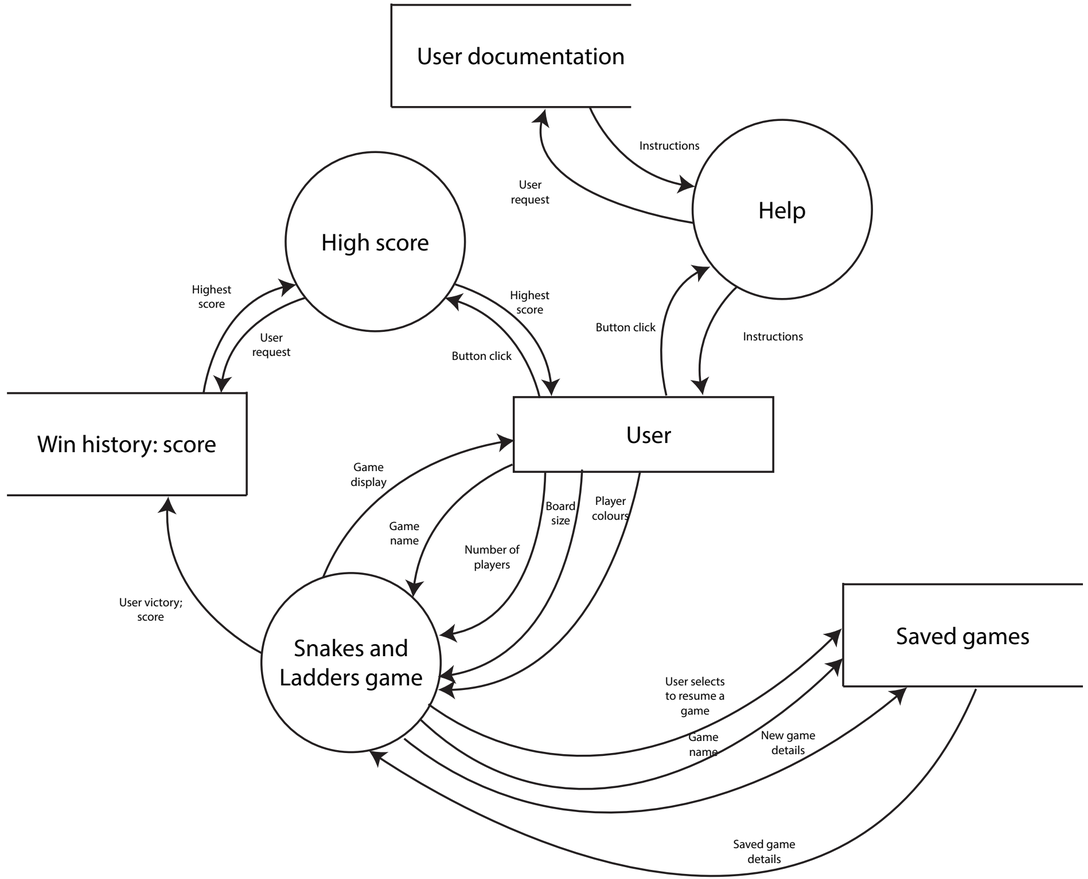


PLANNING AND DESIGNING SOFTWARE SOLUTIONS

DATA FLOW DIAGRAM (DFD)

A DFD is a tool for both system analysis and system design, describing the path that data takes through a system. It assists in understanding the flow of information through a system and is particularly helpful during system design.

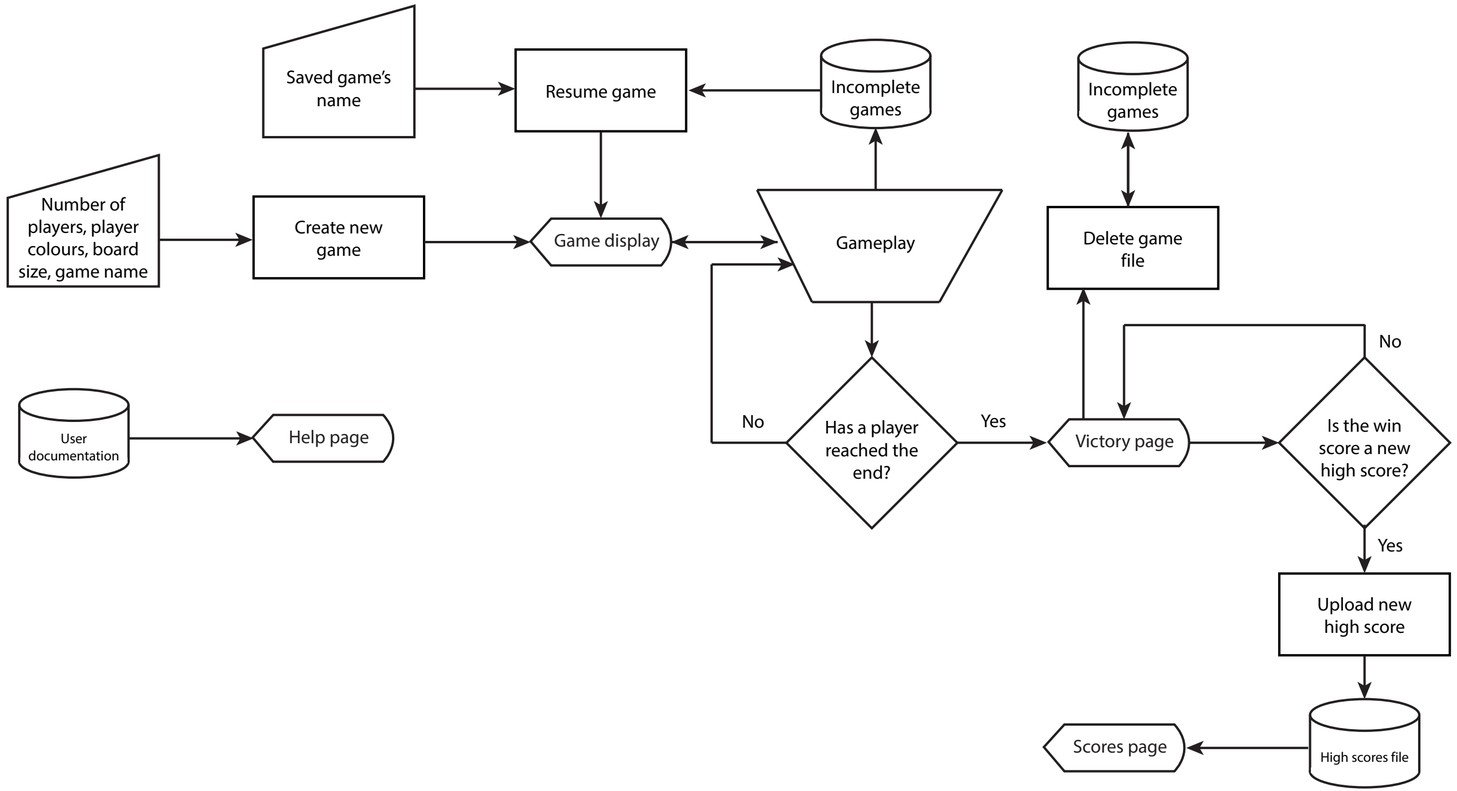
The following DFD describes the flow of data through the Snakes and Ladders system.



SYSTEMS FLOWCHART

Systems flowcharts are used to describe the logic and flow of data between a system's components, including hardware, software and manual components; they describe the interactions between input, processing, output and storage as well as their nature.

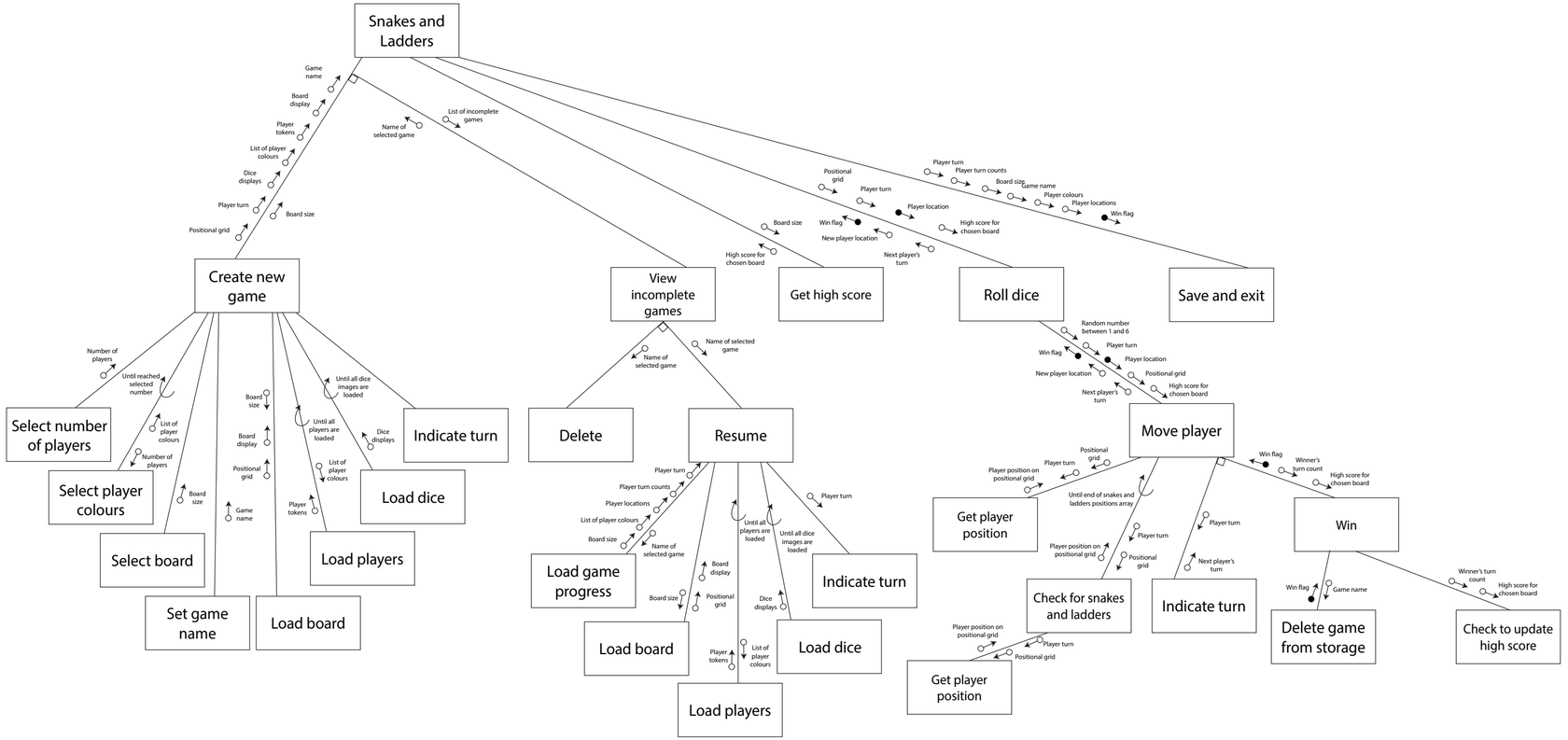
The following systems flowchart shows the interactions between the Snakes and Ladders system components.



STRUCTURE CHART

The primary function of a structure chart is to create a template in preparation for the actual source code. It is used to model the hierarchy of subroutines within a system, with the sequence in which these subroutines take place.

Shown on this page is the structure chart for the Snakes and Ladders game.



DATA DICTIONARY

A data dictionary is the documentation of all variable names or identifiers with their name, data type, length, scope of usage, and its purpose or a description. It is created whilst developing the source code to prevent duplicate identifiers, which assists future maintenance and upgrading of the software solution. A separate one is created for each module, database and file used by the program.

StartMenu

Data Item	Data Type	Format	Description	Example	Validation
gap	Integer	“X”	Gap between two buttons For aesthetics purposes	200	Must be Width of the form divided by 20
underheader	Integer	“X”	The Y-level of beneath the header	100	Must be a positive integer

SelectPlayers

Data Item	Data Type	Format	Description	Example	Validation
Position	Integer	“X”	For positioning the buttons	23	Must be a positive integer
i	Integer	“X”	Incrementer	0	Must be a positive integer

SelectedBoard

Data Item	Data Type	Format	Description	Example	Validation
gap	Integer	“X”	1/10 of the form’s width; used to position buttons	43	Must be a positive integer
SelectedBoard	String	“ZxZ”	The board size selected by the user	“7x7”	Must be either “7x7”, “10x10” or “15x15”

ChooseColours

Data Item	Data Type	Format	Description	Example	Validation
PlayerLimit	Integer	“X”	Global - set from SelectPlayers Indicate how many player colours the user can choose	5	Must be between 2 and 6
W	Integer	“X”	1/6 of the form’s width; used to position controls	49	Must be a positive integer
Starting Limit	Integer	“X”	Starting limit of number of players chosen	1	Must be between 1 and 6
i	Integer	“X”	Incrementer	1	Must be a positive integer
SelectedColours	List of strings	[“Z.png”, “Y.png”...]	Selected player colours	“Red.png” “Pink.png” “Green.png”	Must be out of “Red.png”, “Blue.png”, “Green.png”, “Pink.png”, “Yellow.png” and “Orange.png”

SetGameName

Data Item	Data Type	Format	Description	Example	Validation
AllowedChar	String	“Z”	Space and all alphanumeric characters	“abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 “	Must include all alphanumeric characters and space
GameName	String	“Z”	Set name for the game; for storage purposes	“Game1”	Must be less than 260 characters; all alphanumeric
GameFile	String	“Z.txt”	The text file that will store the game’s details	“Game1.txt”	Must be followed by “.txt”

ViewGames					
Data Item	**Data Type**	**Format**	**Description**	**Example**	**Validation**
GameFile	String	“Z.txt”	The text file of the selected game	“testGame4.txt”	Must be followed by “.txt”
GameName	String	“Z”	The selected game	“testGame4”	Must be less than 260 characters; all alphanumeric
ResumeSelect	Boolean	Y	Resumes the game later if true	1	Must be either true or false
newLines	String	“X”	List of incomplete games without the selected game; overwrites the old list of incomplete games	“Game1 Game2”	Must be more than 0 characters
SavedGamesFile	String	“Z.txt”	The text file where the names of all incomplete games are stored	“!SAVED GAMES!.txt”	Must be “!SAVED GAMES!.txt”
IncompleteGames	Array of strings	[“Z”, “Y”...]	All lines read from “!SAVED GAMES!.txt”	“Game1 Game2”	Must only contain alphanumeric characters

Components					
Data Item	**Data Type**	**Format**	**Description**	**Example**	**Validation**
ScreenWidth	Integer	X	Width of the working area of the display device	1280	Must be a positive integer
ScreenHeight	Integer	X	Height of the working area of the display device	680	Must be a positive integer

HighScoreClass					
Data Item	**Data Type**	**Format**	**Description**	**Example**	**Validation**
HighScore	String	“Z”	Converted into integer - to be compared with UserVictory to determine whether it should be updated by UserVictory and recorded in the text file	7	Must be a positive integer in the form of a string
record	Boolean	Y	Determines which trophy image the victory page should display	1	Must be true or false
UserVictory	Integer	X	The number of turns that the winner took to win the game; converted into string if updating HighScore	10	Must be a positive integer
ScoreFile	String	“Z.txt”	The text file path that stores the high score for the board size	“!HIGH SCORE! 7x7.txt”	Must be followed by “.txt”
NumberOfTurns	List of integers	[X, X, X...]	Stores the number of turns taken by each player	5 4 4	Integers stored must be positive
file	String	“Z.txt”	The text file path to retrieve the high score from	“!HIGH SCORE! 10x10.txt”	Must be followed by “.txt”
HS	String	“Z”	Returns the high score of the selected board size (from the text file path passed from the parameter)	15	Must be a positive integer in the form of a string

Game					
Data Item	**Data Type**	**Format**	**Description**	**Example**	**Validation**
Grid	Dynamic		The positional grid (2D array) that stores the coordinates of all the squares on the board	(100, 230), (120, 230) (100, 210), (120, 210)	Length of row must be the same as length of column, either 7, 10 or 15
lineBreak	String	“”	Represents a line break; used in a text file	“”	Must have no characters
ResumeSelect	Boolean	Y	Resumes the game later if true	0	Must be either true or false
HighScore	String	“Z”	Least number of turns recorded to win a game for the selected board	13	Must be a positive integer
SelectedBoard	String	“Z”	The selected board size	“15x15”	Either “7x7”, “10x10” or “15x15”
BoardSquare	Integer	X	Width of a single square on the board, determined by the chosen board size	30	Must be the result of height of form divided by either 7, 10 or 15
Square1	Point	(X, Y)	The coordinates of the first square on the board	(32, 508)	Coordinates must be positive
ScoreFile	String	“Z.txt”	The text file path that stores the high score of the selected board size; will be used to check the winner’s score with the high score	“!HIGH SCORE! 15x15.txt”	Must end with “.txt”
Finish	Tuple	(X, X)	The position of the finishing square of the board Used to determine whether a player has won or not	(9, 0)	Both integers must be positive
CheckPosArray	Integer	X	Index for accessing the array that stores the positions of the snakes and ladders	2	Must be either 0, 2 or 4
i	Integer	X	Local variable - incrementer	0	Must be a positive integer
j	Integer	X	Local variable - incrementer	3	Must be a positive integer
SelectedColours	List of strings	[“Z.png”, “Y.png”...]	Stores the paths of the selected players’ tokens; Using it with the incrementer inside a loop; ensures the loop loads exactly all the selected players	“Red.png” “Pink.png” “Green.png”	Must be out of “Red.png”, “Blue.png”, “Green.png”, “Pink.png”, “Yellow.png” and “Orange.png”
NumberOfTurns	List of integers	[X, X, X...]	Stores the number of turns taken by each player Incremented as the player finishes their turn	5 4 4	Integers stored must be positive

LocIndex	Tuple	(X, X)	The position of a player on the positional grid Value set by GetPlayerPosition module	(0, 2)	Both integers must be between -1 and 14
playerCounter	Integer	X	Represents which player’s turn it is to move Incremented as the player finishes their turn	4	Must be between 0 and 5
Roll	Integer	X	Random number between 1 and 6 Passed as parameter for MovePlayer module	1	Must be between 1 and 6
LimitX	Integer	X	The number of squares within a row on the board	9	Must be either 6, 9 or 14
LimitY	Integer	X	The number of squares within a column on the board	14	Must be either 6, 9 or 15
exceedTotal	Integer	X	The number of squares that the player will go over the finishing square by (if the player is on the final square and hasn’t stopped)	4	Must be between 1 and 5
GameFile	String	“Z.txt”	The text file path that stores the game details of the current game	“Game1.txt”	Must end with “.txt”
colour	String	“Z.png”	Used in a foreach statement with SelectedColours All the colours of the players are saved into the text file of this game	“Pink.png”	Must be either of “Red.png”, “Blue.png”, “Green.png”, “Pink.png”, “Yellow.png” or “Orange.png”
turns	Integer	X	Used in a foreach statement with NumberOfTurns The number of turns that each player has taken so far are saved into the text file	2	Must be a positive integer
SavedGamesFile	String	“Z.txt”	The text file path where the names of all incomplete games are stored	“!SAVED GAMES.txt!”	Must be “!SAVED GAMES!.txt”
GameName	String	“Z”	The name of this game Added to the incomplete games text file	“game6”	Must be less than 260 characters; all alphanumeric
lines	Array of strings	[“Z”, “Y”...]	All the lines from the SavedGamesFile text file Is overwritten so that there are no duplicates of the same game	“Game1” “Game2” “test”	All strings inside must only contain alphanumeric characters
line	String	“Z”	Local variable - Represents a line in the text file that’s read	“7x7”	Must be alphanumeric characters
coord	String	“Z”	Used in switch case: indicator for positioning either the x or y coordinate of a player token	“x”	Must be either “x” or “y”

GameData					
Data Item	**Data Type**	**Format**	**Description**	**Example**	**Validation**
ResumeSelect	Boolean	Y	Resumes the game later if true	1	Must be either true or false
PlayerLimit	Integer	“X”	Global - set from SelectPlayers Indicate how many player colours the user can choose	5	Must be between 2 and 6
GameFile	String	“Z.txt”	The text file of the selected game	“testGame4.txt”	Must be followed by “.txt”
GameName	String	“Z”	The selected game	“testGame4”	Must be less than 260 characters; all alphanumeric
SavedGamesFile	String	“Z.txt”	The text file where the names of all incomplete games are stored	“!SAVED GAMES!.txt”	Must be “!SAVED GAMES!.txt”
IncompleteGames	Array of strings	[“Z”, “Y”…]	All lines read from “!SAVED GAMES!.txt”	“Game1 Game2”	Must only contain alphanumeric characters
SelectedColours	List of strings	[“Z.png”, “Y.png”…]	Selected player colours	“Red.png” “Pink.png” “Green.png”	Must be out of “Red.png”, “Blue.png”, “Green.png”, “Pink.png”, “Yellow.png” and “Orange.png”
playerCounter	Integer	X	Represents which player’s turn it is to move	4	Must be between 0 and 5

BoardClass					
Data Item	**Data Type**	**Format**	**Description**	**Example**	**Validation**
Finish	Tuple	(X, X)	The position of the finishing square of the board Used to determine whether a player has won or not	(9, 0)	Both integers must be positive
Square1	Point	(X, Y)	The coordinates of the first square on the board	(32, 508)	Coordinates must be positive
BoardSquare	Integer	X	Width of a single square on the board, determined by the chosen board size	30	Must be the result of height of form divided by either 7, 10 or 15
SelectedBoard	String	“Z”	The selected board size	“15x15”	Either “7x7”, “10x10” or “15x15”
CheckPosArray	Integer	X	Index for accessing the array that stores the positions of the snakes and ladders	2	Must be either 0, 2 or 4
ChangePos7	Array of tuples	[(X, X), (Y, Y)…]	Stores the tuple positions that have snakes or ladders on the 7x7 board	(2, 2) (5, 4)	Tuples inside must be integers from 0 to 6
FinalPos7	Array of tuples	[(X, X), (Y, Y)…]	Stores the new tuple positions (after landing on a snake or ladder) on the 7x7 board	(5, 1) (6, 5)	Tuples inside must be integers from 0 to 6
ChangePos10	Array of tuples	[(X, X), (Y, Y)…]	Stores the tuple positions that have snakes or ladders on the 10x10 board	(5, 6) (8, 1)	Tuples inside must be integers from 0 to 9
FinalPos10	Array of tuples	[(X, X), (Y, Y)…]	Stores the new tuple positions (after landing on a snake or ladder) on the 10x10 board	(1, 8) (3, 5)	Tuples inside must be integers from 0 to 9
ChangePos15	Array of tuples	[(X, X), (Y, Y)…]	Stores the tuple positions that have snakes or ladders on the 15x15 board	(10, 8) (11, 13)	Tuples inside must be integers from 0 to 14
FinalPos15	Array of tuples	[(X, X), (Y, Y)…]	Stores the new tuple positions (after landing on a snake or ladder) on the 15x15 board	(10, 6) (2, 9)	Tuples inside must be integers from 0 to 14
SnakesLadders	List of arrays of tuples	[X, Y...]	Stores the arrays of tuples that contain the snakes and ladders positions	ChangePos7 FinishPos7 …	Must contain 6 arrays

ALGORITHMS

Algorithms are methods of solving a problem; they describe the processing steps that transform the inputs into the outputs within a solution. The purpose of creating them is to explain the logic of the solution

The following algorithms describe the Snakes and Ladders system in pseudocode.

```
BEGIN Game.MAIN METHOD
    Get GameData.ResumeSelect
    IF GameData.ResumeSelect = true THEN
        ResumeGame
    ELSE
        LoadBoard
        LoadPlayers
    ENDIF

    LoadDice
    IndicateTurn
    Get HighScoreClass.HighScore, GameData.ScoreFile
    Let HighScoreClass.HighScore = HighScoreClass.GetHighScore(GameData.ScoreFile)

    Set RollButton.Click to RollButton_Click
    Set Exit.Click to _SaveGame
    Set Return.Click to _SaveGame
END Game.MAIN METHOD
```

```
BEGIN ResumeGame
    Get GameData.GameFile, BoardClass.SelectedBoard, lineBreak, GameData.SelectedColours, HighScoreClass.NumberOfTurns, GameData.playerCounter
    Open GameData.GameFile for input

    Read BoardClass.SelectedBoard from GameData.GameFile
    LoadBoard

    Read lineBreak from GameData.GameFile

    Let i = 0
    WHILE Read from GameData.GameFile <> lineBreak
        Read GameData.SelectedColours (i) from GameData.GameFile
        Let i = i + 1
    ENDWHILE
    LoadPlayers

    Let i = 0
    Let coord = "x"
    WHILE Read from GameData.GameFile <> lineBreak
        CASEWHERE coord is
            "x":
                Read X-coordinate of GameData.PlayerList (i) from GameData.GameFile
                Let coord = "y"
            "y":
                Read Y-coordinate of GameData.PlayerList (i) from GameData.GameFile
                Let coord = "x"
                Let i = i + 1
            OTHERWISE:
        ENDCASE
    ENDWHILE
    Let i = 0
    WHILE Read from GameData.GameFile <> lineBreak
        Read HighScoreClass.NumberOfTurns (i) from GameData.GameFile
        Let i = i + 1
    ENDWHILE

    Read GameData.playerCounter from GameData.GameFile
    Close GameData.GameFile
END ResumeGame
```

```
BEGIN LoadBoard
    Get BoardClass.SelectedBoard, Height of Form
    CASEWHERE BoardClass.SelectedBoard is
        "7x7":
            Set background image to "7x7.png"
            Set BoardClass.BoardSquare to (Height of Form/7)
            Set BoardClass.Square1 to (BoardClass.DefaultBoard(Game).Location.X, Height of Form – BoardClass.BoardSquare)
            Set GameData.ScoreFile to "!HIGH SCORE 7x7!.txt"
            Set Grid to BoardClass.BoardGrid(7)
            Set BoardClass.Finish to (6, 6)
            Set BoardClass.CheckPosArray to 0
        "10x10":
            Set background image to "10x10.png"
            Set BoardClass.BoardSquare to (Height of Form/10)
            Set BoardClass.Square1 to (BoardClass.DefaultBoard(Game).Location.X, Height of Form – BoardClass.BoardSquare)
            Set GameData.ScoreFile to "!HIGH SCORE 10x10!.txt"
            Set Grid to BoardClass.BoardGrid(10)
            Set BoardClass.Finish to (9, 0)
            Set BoardClass.CheckPosArray to 2
        "15x15":
            Set background image to "15x15.png"
            Set BoardClass.BoardSquare to (Height of Form/15)
            Set BoardClass.Square1 to (BoardClass.DefaultBoard(Game).Location.X, Height of Form – BoardClass.BoardSquare)
            Set GameData.ScoreFile to "!HIGH SCORE 15x15!.txt"
            Set Grid to BoardClass.BoardGrid(15)
            Set BoardClass.Finish to (14, 14)
            Set BoardClass.CheckPosArray to 4
        OTHERWISE:
    ENDCASE
END LoadBoard
```

```
BEGIN LoadPlayers
        Let i = 0
        Get GameData.SelectedColours, GameData.PlayerList, PlayerTurns, BoardClass.BoardSquare, LeftBG, HighScoreClass.NumberOfTurns
        WHILE i < GameData.SelectedColours.Length
                Let GameData.PlayerList (i) = Components.PicBox(GameData.SelectedColours (i), BoardClass.BoardSquare/2, BoardClass.BoardSquare/2)
                Let GameData.PlayerList (i).Location = (BoardClass.DefaultBoard(Game).Location.X – BoardClass.BoardSquare – i*BoardClass.BoardSquare/2, Height of Form – BoardClass.BoardSquare)
                Add GameData.PlayerList (i) to Game

                Let PlayerTurns (i) = Components.PicBox(GameData.SelectedColours (i), BoardClass.BoardSquare, BoardClass.BoardSquare)
                Let PlayerTurns (i).Location = (0, i*BoardClass.BoardSquare)
                Add PlayerTurns (i) to LeftBG

                Let HighScoreClass.NumberOfTurns (i) = 0
                Let i = i + 1
        ENDWHILE
END LoadPlayers


BEGIN LoadDice
        Get GameData.DiceStorage, BoardClass.BoardSquare, Height of Form, RightBG
        Let i = 0
        WHILE i < Length of GameData.DiceStorage
                Let GameData.DiceStorage (i).Size = (BoardClass.DefaultBoard(Game).Location.X/2, BoardClass.DefaultBoard(Game).Location.X/2)
                Let GameData.DiceStorage (i).Visible = false
                Let GameData.DiceStorage (i).Location = (BoardClass.BoardSquare, Height of Form – BoardClass.BoardSquare – Height of GameData.DiceStorage (i))
                Let i = i + 1
        ENDWHILE
        Add GameData.DiceStorage to RightBG
END LoadDice
```



```
BEGIN IndicateTurn
        Get GameData.playerCounter, GameData.PlayerList, PlayerArrow, BoardClass.BoardSquare, PlayerTurns
        IF GameData.playerCounter = Length of GameData.PlayerList THEN
                Let GameData.playerCounter = 0
        ENDIF
        Let PlayerArrow.Location = (BoardClass.BoardSquare, PlayerTurns(GameData.playerCounter).Location.Y)
END IndicateTurn


BEGIN GetPlayerPosition
        Let LocIndex = (0, -1)
        Get Grid, GameData.PlayerList, GameData.playerCounter

        FOR i = 0 TO Length of Grid.X STEP 1
                FOR j = 0 TO Length of Grid.Y STEP 1
                        IF GameData.PlayerList(GameData.playerCounter).Location = Grid (i, j) THEN
                                Let LocIndex = (i, j)
                        ENDIF
                NEXT j
        NEXT i

        RETURN LocIndex
END GetPlayerPosition
```

```
BEGIN MovePlayer (Roll)
        Get Grid
        Let LimitX = Length of Grid.X - 1
        Let LimitY = Length of Grid.Y – 1
        Let LocIndex = GetPlayerPosition

        WHILE Roll > 0
                IF (X-Value of LocIndex) / 2 Remainder 0 THEN
                        IF X-Value of LocIndex = LimitX AND Y-Value of LocIndex = LimitY THEN
                                Let exceedTotal = Y-Value of LocIndex + Roll
                                Let Y-Value of LocIndex = LimitY – (exceedTotal – LimitY)
                                Let Roll = 0
                        ELSE IF Y-Value of LocIndex = LimitY THEN
                                Let X-Value of LocIndex = X-Value of LocIndex + 1
                                Let Y-Value of LocIndex = Y-Value of LocIndex  - Roll + 1
                                Let Roll = 0
                        ELSE
                                Let Y-Value of LocIndex = Y-Value of LocIndex + 1
                                Let Roll = Roll – 1
                        ENDIF
                ELSE
                        IF X-Value of LocIndex = LimitX AND Y-Value of LocIndex = 0 THEN
                                Let exceedTotal = Y-Value of LocIndex – Roll
                                Let Y-Value of LocIndex = – exceedTotal
                                Let Roll = 0
                        ELSE IF Y-Value of LocIndex = 0 THEN
                                Let X-Value of LocIndex = X-Value of LocIndex + 1
                                Let Y-Value of LocIndex = Roll – 1
                                Let Roll = 0
                        ELSE
                                Let Y-Value of LocIndex = Y-Value of LocIndex – 1
                                Let Roll = Roll – 1
                        ENDIF
                ENDIF
        ENDWHILE

        Let GameData.PlayerList (GameData.playerCounter).Location = Grid (X-Value of LocIndex, Y-Value of LocIndex)
        CheckSnakesLadders

        Let HighScoreClass.NumberOfTurns (GameData.playerCounter) = HighScoreClass.NumberOfTurns (GameData.playerCounter) + 1

        Get BoardClass.Finish
        IF LocIndex = BoardClass.Finish THEN
                Show Victory Form
        ELSE
                Let GameData.playerCounter = GameData.playerCounter + 1
                Enable RollButton
                IndicateTurn
        ENDIF
END MovePlayer
```

```
BEGIN CheckSnakesLadders
        Get BoardClass.SnakesLadders, BoardClass.CheckPosArray, GameData.PlayerList, GameData.playerCounter, Grid
        Let LocIndex = GetPlayerPosition
        Let ChangePos = BoardClass.SnakesLadders (BoardClass.CheckPosArray)
        Let BoardClass.FinishPos = BoardClass.SnakesLadders (BoardClass.CheckPosArray + 1)

        FOR i = 0 TO Length of ChangePos STEP 1
                IF LocIndex = ChangePos (i)
                        Let LocIndex = BoardClass.FinishPos (i)
                        Let GameData.PlayerList (GameData.playerCounter).Location  = Grid (LocIndex.X, LocIndex.Y)
                ENDIF
        NEXT i
END CheckSnakesLadders
```

```
BEGIN RollButton_Click
        Get RollButton, GameData.DiceStorage
        Disable RollButton
        Let Roll = Random(5) + 1
        FOR i = 0 to 6 STEP 1
                Let GameData.DiceStorage (i).Visible = true
                Let GameData.DiceStorage (i).Visible = false
        NEXT i
        Let GameData.DiceStorage (Roll – 1).Visible = true
        MovePlayer (Roll)
END RollButton_Click
```

```
BEGIN _SaveGame
        Get GameData.GameFile, BoardClass.SelectedBoard, lineBreak, GameData.SelectedColours
        Get GameData.PlayerList, HighScoreClass.NumberOfTurns, GameData.playerCounter
        Open GameData.GameFile for output

        Write GameData.GameFile from BoardClass.SelectedBoard
        Write GameData.GameFile from lineBreak

        Let i = 0
        WHILE i < Length of GameData.SelectedColours
                Write GameData.GameFile from GameData.SelectedColours (i)
                Let i = i + 1
        ENDWHILE
        Write GameData.GameFile from lineBreak

        Let i = 0
        WHILE i < Length of GameData.PlayerList
                Write GameData.GameFile from X-coordinate of GameData.PlayerList (i)
                Write GameData.GameFile from Y-coordinate of GameData.PlayerList (i)
                Let i = i + 1
        ENDWHILE
        Write GameData.GameFile from lineBreak

        Let i = 0
        WHILE i < Length of HighScoreClass.NumberOfTurns
                Write GameData.GameFile from HighScoreClass.NumberOfTurns (i)
                Let i = i + 1
        ENDWHILE
        Write GameData.GameFile from lineBreak

        Write GameData.GameFile from GameData.playerCounter

        Close GameData.GameFile

        Get GameData.SavedGamesFile, GameData.GameName
        Open GameData.SavedGamesFile for output

        Write GameData.SavedGamesFile from GameData.GameName
        Close GameData.SavedGamesFile

        Get lines
        Extract unique characters from GameData.SavedGamesFile into lines
        Set GameData.SavedGamesFile to lines
END _SaveGame
```

```
BEGIN PicBox (file, W, H)
        Let picturebox.Image = file
        Let picturebox.Size = (W, H)
        RETURN picturebox
END PicBox

BEGIN DefaultBoard (form)
        Let board.Size = (form.Height, form.Height)
        Let board.Location = (FormCentre(form).X – form.Height/2, 0)
        RETURN board
END DefaultBoard
```

```
BEGIN BoardGrid (size)
        Let Grid = Grid (size, size)
        Get BoardClass.Square1, BoardClass.BoardSquare
        Let Position = BoardClass.Square1

        Let Row = 0
        WHILE Row < Length of Grid.X
                IF Row / 2 Remainder 0 THEN
                        Let Col = 0
                        WHILE Col < Length of Y of Grid
                                Let Grid (Row, Col) = (Position.X + Col*BoardClass.BoardSquare, Position.Y)
                                Let Col = Col + 1
                        ENDWHILE
                        Let Position = (Position.X, Position.Y – BoardClass.BoardSquare)
                ELSE
                        Let Col = Length of Grid.Y – 1
                        WHILE Col >= 0
                                Let Grid (Row, Col) = (Position.X + Col*BoardClass.BoardSquare, Position.Y)
                                Let Col = Col – 1
                        ENDWHILE
                        Let Position = (Position.X, Position.Y – BoardClass.BoardSquare)
                ENDIF
        ENDWHILE
        RETURN Grid
END BoardGrid
```

DESK CHECKING

Desk checking is the primary technique for checking algorithms, by using test data. The process of desk checking is the evaluation of each statement in the algorithm with the results written on a table, which has a column for each identifier used within the algorithm.

The following spreads display the desk checks for the algorithms for the Snakes and Ladders solution.

ResumeGame

GameData.GameFile	i	SelectedColours(i)	coord	PlayerList(i).Location	HighScoreClass.NumberOfTurns(i)	playerCounter
“Red.png”	0	“Red.png”				
“Blue.png”	1	“Blue.png”				
“Yellow.png”	2	“Yellow.png”				
lineBreak	3					
203	0		“x”	(203, 0)		
102	0		“y”	(203, 102)		
314	1		“x”	(314, 0)		
319	1		“y”	(314, 319)		
809	2		“x”	(809, 0)		
358	2		“y”	(809, 358)		
lineBreak	3					
2	0				2	
2	1				2	
1	2				1	
lineBreak	3					
2	3					2

LoadBoard

BoardClass.SelectedBoard	background image	Height of Form	BoardClass.BoardSquare	BoardClass.DefaultBoard.Location	BoardClass.Square1	GameData.ScoreFile	Grid	BoardClass.Finish	BoardClass.CheckPosArray
“7x7”	“7x7.png”	1000	143	(560, 0)	(560, 857)	“!HIGH SCORE 7x7!.txt”	BoardClass.BoardGrid(7)	(6, 6)	0
“10x10”	“10x10.png”	1000	100	(560, 0)	(560, 900)	“!HIGH SCORE 10x10!.txt”	BoardClass.BoardGrid(10)	(9, 0)	2
“15x15”	“15x15.png”	1000	67	(560, 0)	(560, 933)	“!HIGH SCORE 15x15!.txt”	BoardClass.BoardGrid(15)	(14, 14)	4

LoadPlayers

i	GameData.SelectedColours.Length	Height of Form	GameData.SelectedColours(i)	GameData.BoardSquare	GameData.PlayerList(i)	BoardClass.DefaultBoard(Game).Location	GameData.PlayerList(i).Location	PlayerTurns(i)	PlayerTurns(i).Location	HighScoreClass.NumberOfTurns(i)
0	3	1200	“Orange.png”	120	Components.PicBox(“Orange.png”, 60, 60)	(760, 0)	(640, 1080)	Components.PicBox(“Orange.png”, 60, 60)	(0, 0)	0
1	3	1200	“Red.png”	120	Components.PicBox(“Red.png”, 60, 60)	(760, 0)	(580, 1080)	Components.PicBox(“Red.png”, 60, 60)	(0, 120)	0
2	3	1200	“Yellow.png”	120	Components.PicBox(“Yellow.png”, 60, 60)	(760, 0)	(520, 1080)	Components.PicBox(“Yellow.png”, 60, 60)	(0, 240)	0
3	3	1200		120		(760, 0)				

LoadDice							
i	**Length of GameData. DiceStorage**	**Height of Form**	**BoardClass. DefaultBoard(Game). Location**	**BoardClass. BoardSquare**	**GameData. DiceStorage(i). Size**	**GameData. DiceStorage(i). Visible**	**GameData. DiceStorage(i). Location**
0	6	1300	(300, 0)	130	(150, 150)	0	(130, 1020)
1	6	1300	(300, 0)	130	(150, 150)	0	(130, 1020)
2	6	1300	(300, 0)	130	(150, 150)	0	(130, 1020)
3	6	1300	(300, 0)	130	(150, 150)	0	(130, 1020)
4	6	1300	(300, 0)	130	(150, 150)	0	(130, 1020)
5	6	1300	(300, 0)	130	(150, 150)	0	(130, 1020)
6	6	1300	(300, 0)	130			

IndicateTurn				
Length of GameData. PlayerList	**Game. playerCounter**	**BoardClass. BoardSquare**	**PlayerTurns (GameData. playerCounter). Location**	**PlayerArrow. Location**
3	3	140	(0, 230)	(140, 230)
3	0	140	(0, 230)	(140, 230)
3	0	140	(0, 0)	(140, 0)

GetPlayerPosition							
LocIndex	**GameData. PlayerList (GameData. playerCounter). Location**	**GameData. playerCounter**	**Length of Grid.X**	**Length of Grid.Y**	**i**	**j**	**Grid (i, j)**
(0, -1)	(150, 250)	0	7	7	0	0	(100, 300)
(0, -1)	(150, 250)	0	7	7	0	1	(150, 300)
(0, -1)	(150, 250)	0	7	7	0	2	(200, 300)
(0, -1)	(150, 250)	0	7	7	0	3	(250, 300)
(0, -1)	(150, 250)	0	7	7	0	4	(300, 300)
(0, -1)	(150, 250)	0	7	7	0	5	(350, 300)
(0, -1)	(150, 250)	0	7	7	0	6	(400, 300)
(0, -1)	(150, 250)	0	7	7	1	0	(100, 250)
(1, 1)	(150, 250)	0	7	7	1	1	(150, 250)
(1, 1)	(150, 250)	0	7	7	1	2	(200, 250)
Until i = 7 and j = 7							

MovePlayer									
Roll	**Length of Grid.X**	**Length of Grid.Y**	**LimitX**	**LimitY**	**LocIndex**	**exceedTotal**	**GameData. playerCounter**	**HighScoreClass. NumberOfTurns (GameData. playerCounter)**	**BoardClass. Finish**
3	10	10	9	9	(0, -1)		1	0	
3	10	10	9	9	(0, 0)		1	0	
2	10	10	9	9	(0, 1)		1	0	
1	10	10	9	9	(0, 2)		1	0	
0	10	10	9	9	(0, 2)		1	1	
0	10	10	9	9	(0, 2)		1	1	(9, 0)
0	10	10	9	9	(0, 2)		2	1	(9, 0)

_SaveGame										
i	**BoardClass. SelectedBoard**	**lineBreak**	**Length of GameData. SelectedColours**	**GameData. SelectedCo-lours(i)**	**Length of GameData. PlayerList**	**GameData. PlayerList (i). Location**	**Length of HighScore-Class. NumberOfTurns**	**HighScoreClass. NumberOfTurns(i)**	**GameData. playerCounter**	**GameData. GameFile**
	“7x7”	“”	3		3		3		2	“7x7 “
0	“7x7”	“”	3	“Red.png”	3	(150, 250)	3	1	2	“7x7 Red.png”
1	“7x7”	“”	3	“Green.png”	3	(200, 250)	3	1	2	“7x7 Red.png Green.png”
2	“7x7”	“”	3	“Blue.png”	3	(100, 200)	3	0	2	“7x7 Red.png Green.png Blue.png”
3	“7x7”	“”	3		3		3		2	“7x7 Red.png Green.png Blue.png “
0	“7x7”	“”	3	“Red.png”	3	(150, 250)	3	1	2	“7x7 Red.png Green.png Blue.png 150 250”
1	“7x7”	“”	3	“Green.png”	3	(200, 250)	3	1	2	“7x7 Red.png Green.png Blue.png 150 250 200 250”
2	“7x7”	“”	3	“Blue.png”	3	(100, 200)	3	0	2	“7x7 Red.png Green.png Blue.png 150 250 200 250 100 200”
3	“7x7”	“”	3		3		3		2	“7x7 Red.png Green.png Blue.png 150 250 200 250 100 200 “
0	“7x7”	“”	3	“Red.png”	3	(150, 250)	3	1	2	“7x7 Red.png Green.png Blue.png 150 250 200 250 100 200 1”

1	“7x7”	“”	3	“Green.png”	3	(200, 250)	3	1	2	“7x7 Red.png Green.png Blue.png 150 250 200 250 100 200 1 1”
2	“7x7”	“”	3	“Blue.png”	3	(100, 200)	3	0	2	“7x7 Red.png Green.png Blue.png 150 250 200 250 100 200 1 1 0”

3	“7x7”	“”	3		3		3		2	“7x7 Red.png Green.png Blue.png 150 250 200 250 100 200 1 1 0 “
3	“7x7”	“”	3		3		3		2	“7x7 Red.png Green.png Blue.png 150 250 200 250 100 200 1 1 0 2”

RollButton_Click			
Roll	**i**	**GameData. DiceStorage(i). Visible**	**GameData. DiceStorage (Roll - 1). Visible**
4	0	1	
4	0	0	
4	1	1	
4	1	0	
4	2	1	
4	2	0	
4	3	1	
4	3	0	
4	4	1	
4	4	0	
4	5	1	
4	5	0	
4	6		1

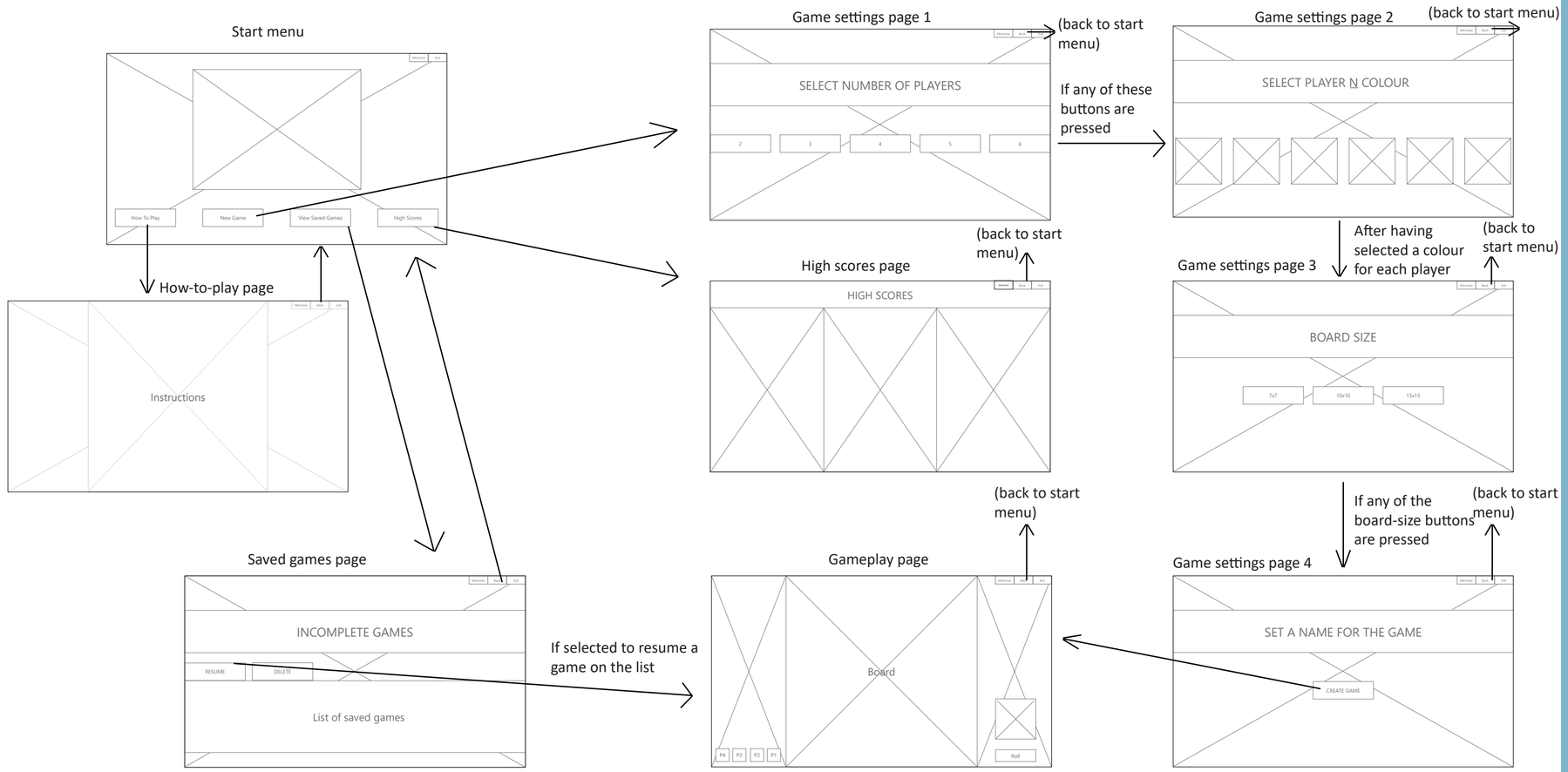
CheckSnakesLadders							
LocIndex	**i**	**Length of ChangePos**	**ChangePos(i)**	**BoardClass. FinishPos(i)**	**GameData. playerCounter**	**Grid (LocIndex.X, LocIndex.Y)**	**GameData. PlayerList (GameData. playerCounter). Location**
(0, 0)	0	7	(0, 0)	(1, 2)	1	(230, 680)	(230, 680)
(1, 2)	0	7	(0, 0)	(1, 2)	1	(230, 680)	(230, 680)
(1, 2)	1	7	(0, 0)	(1, 2)	1	(330, 480)	(330, 480)

BoardGrid							
Row	**Col**	**BoardClass. Square1**	**Position**	**Length of Grid.X**	**Length of Grid.Y**	**BoardClass. BoardSquare**	**Grid (Row, Col)**
0		(50, 400)	(50, 400)	7	7	50	
0	0	(50, 400)	(50, 400)	7	7	50	(50, 400)
0	1	(50, 400)	(50, 400)	7	7	50	(100, 400)
0	2	(50, 400)	(50, 400)	7	7	50	(150, 400)
0	3	(50, 400)	(50, 400)	7	7	50	(200, 400)
until Col = 7							
0	7	(50, 400)	(50, 350)	7	7	50	
1	6	(50, 400)	(50, 350)	7	7	50	(350, 350)
1	5	(50, 400)	(50, 350)	7	7	50	(300, 350)
1	4	(50, 400)	(50, 350)	7	7	50	(250, 350)
until Col < 0							
until Row = 7							

STORYBOARD

A storyboard shows the various interfaces in a system and the links between them; it identifies the purpose, contents and design elements of the system.

The following storyboard presents the interfaces in the Snakes and Ladders system.



IMPLEMENTING SOFTWARE SOLUTIONS

LOGBOOK

Logbooks are used to document the progress of a project, with each entry comprising:

- date
- description of the progress (or lack thereof) made since the last entry
- tasks achieved
- stumbling blocks
- details of possible approaches for upcoming tasks
- references to resources used

03/06

Tasks Achieved

- Creating controls for setting a new game
- Creating headings

Stumbling Blocks

- Learning how to use a panel to lay things out

07/06

Tasks Achieved

- Designed boards (only the layout)
- Designed players
- Method for board selection
- Method for loading players

12/06

Tasks Achieved

- Made a method that returns a 2D-Array that stores the positions of the selected board

Stumbling blocks

Resources

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/multidimensional-arrays

https://stackoverflow.com/questions/12826760/printing-2d-array-in-matrix-format

13/06

Tasks Achieved

- Made the players move
- Made the player icon's background transparent to the board

Stumbling blocks

- Going out of bounds on the 5x5 board when the player is moving up by 2 y-levels

Resources

https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.control.backgroundimage?view=net-5.0

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-tuples

https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.application.exit?view=net-5.0

19/06
Tasks Achieved

- Coded the save game function
- Coded the resume game function

Stumbling blocks

- (all necessary game data were saved onto one text file but the different parts are separated by line breaks) Retrieving different data and resetting the variables for the game by the line breaks
- Resetting each player's positions (according to the format it's saved as in the text file)

Resources
https://docs.microsoft.com/en-us/dotnet/api/system.io.streamreader.readtoend?view=net-5.0

https://docs.microsoft.com/en-us/dotnet/desktop/winforms/controls/how-to-position-controls-on-windows-forms?view=netframeworkdesktop-4.8

https://stackoverflow.com/questions/14978095/stream-reader-readline-detect-newline-characters

20/06
Tasks Achieved

- Allowing the user to select from a saved game

Stumbling blocks

- Coding the GUI for this method

Resources
https://stackoverflow.com/questions/2943330/how-to-make-listboxs-texts-center-aligned-in-desktop-application-using-c-net

21/06
Tasks Achieved

- Coded the HighScore function - recording and checking for the lowest number of dice rolls in a game
- Allowing the HighScore function to run across 3 different boards
- Allowing the user to only enter alphanumerical characters for the game name (a file cannot be named with certain characters)
- Coded the function that allows the user to delete a saved game

Stumbling blocks

Resources
https://stackoverflow.com/questions/33018299/how-to-prevent-users-from-typing-special-characters-in-textbox

https://stackoverflow.com/questions/668907/how-to-delete-a-line-from-a-text-file-in-c?noredirect=1&lq=1

23/06
Tasks Achieved

- User can delete the file of a saved game from the PC

26/06
Tasks Achieved

- Finished designing the boards, including the Snakes and Ladders positions
- Recorded the snakes and ladders' relative positions in arrays
- Polished GUI a bit using panels
- Coded method that updates the player's turn e.g. It's Red's turn!
- Coded function that removes duplicate games -> saved games with the same name will overwrite each other now

Stumbling blocks

- Struggled to make the controls' background transparent (with the technique of parenting controls to each other)
- The 'delete' saved game function still needs debugging, it currently cannot update the 'savedgames' text file when the user deletes multiple things in one run (cannot refresh every time)

Resources
https://stackoverflow.com/questions/1245500/remove-duplicate-lines-from-text-file

28/06
Tasks Achieved

- Coded GUI for HighScores page
- Debugged a HighScore method: Each player's number of turns were messed up when resuming and saving the game
- Created custom button class

Stumbling blocks

Resources
https://www.instructables.com/C-Custom-Button-Control-in-10mins/

30/06
Tasks Achieved

- Designed 'How to play' page
- Improved button designs; added hover effects

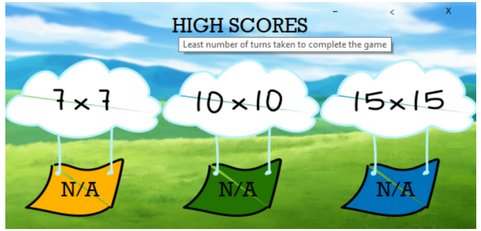
Resources
https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.cursors.hand?view=net-5.0

USER DOCUMENTATION

The purpose of user documentation is to teach the users about the operation of aspects of the software product.

My solution provides an installation guide, as well as a short ‘How To Play’ page and tool tips within the application.

Tool tips: small windows containing a brief description of the current screen element that the mouse is hovering over.



‘How To Play’ page

Installation guide (comes in a .txt file)

```
Snakes and Ladders INSTALLATION GUIDE

1. Download the "App" ZIP file.
2. Click "Extract All" to select a destination and extract the files.
3. Open the "App" folder.
4. Run the "Snakes and Ladders" .exe file.
```

TESTING AND EVALUATING THE SOFTWARE SOLUTION

BETA TESTING

Beta testing is a test method used to evaluate the level of customer satisfaction with the software product by having it validated by the end users.

The following logs detail the process of end-user testing.

14/07

- Need to start beta-testing

Steps required

- Send out beta program for end-user testing
- Construct survey to obtain feedback
- Receive feedback
- Graph and tabulate feedback
- Evaluate feedback

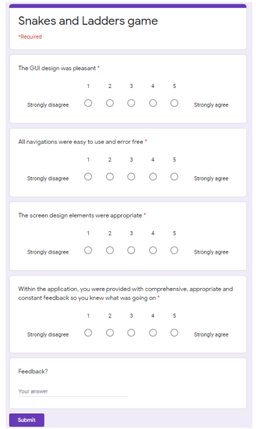
15/07

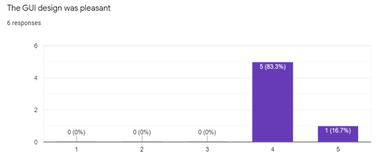
Tasks Achieved

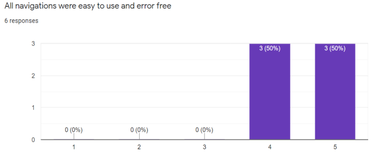
- Constructed survey for end-user testing
- Survey sent out to friends

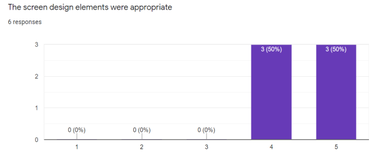
Next up

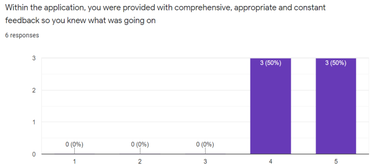
- Retain results
- Graph and tabulate results
- Hopefully by the end of next week











		Results				
		GUI	**Navigations**	**Screen design elements**	**In-game feedback**	**Comments**
Response	**User1**	4/5	5/5	5/5	5/5	The screens are a bit glitchy
	User2	4/5	4/5	5/5	4/5	no sound :l
	User3	4/5	4/5	4/5	4/5	
	User4	4/5	4/5	4/5	4/5	Showing the players moving step by step would've been nice
	User5	4/5	5/5	4/5	5/5	
	User6	5/5	5/5	5/5	5/5	I like the trophy :DD

22/07

Tasks Achieved

- Survey responses received from 6 people
- Results are graphed and tabulated
- Results are evaluated concerning end-user satisfaction and identifying suggested changes/improvements

Overall, the end-users provided positive feedback and it can be inferred that their experience with the game was mostly pleasant.

However, specific areas for improvement include screen glitches, providing more user feedback in game such as sound effects and player-movement animations.

From the graphed results, the GUI design received the worst feedback, therefore it should be an area of focus for improvements.

EVALUATION OF OBJECTIVES

The aim of any newly developed software solution is to meet its original requirements and design specifications. Evaluating the solution against these specifications ensures that the user requirements are being met.

Therefore, the objectives that vwere set when defining and understanding the problem are evaluated.

Objective	Action
The application must adhere to current trends in user interface development; must be aesthetically pleasing and family-friendly.	✓
All navigation must be appropriate and error free.	✓
The selection of appropriate interface objects and graphics must be consistent and professional in nature	✓
The application must provide users with appropriate and relevant feedback. This must assist the user to feel in control of the application and must include aspects like alerts, notifications and labels.	✓
Appropriate control structures must be implemented into the program, ranging from sequence, selection and repetition.	✓
Appropriate data structures must be implemented into the program. This should be either arrays or records.	✓
A 2-Dimensional array must be used.	✓
The user must be able to save and resume incomplete games.	✓
The game must have at least 3 different board set-ups.	✓
The game must have a dice animation.	✓
The game must be capable of recording and updating the high score for each board type. This must be accessible to the user as well.	✓
The solution should be a functional Snakes and Ladders game in considering the initial objectives and free of errors.	✓

All my objectives were achieved, meaning that my solution should accurately reflect the product needed based on the original requirements.

However, improvements can still be made, particularly with the GUI. Although I evaluated that the first objective has been achieved - adhering to current trends in UI development, being aesthetically pleasing, many screen design elements can still be improved, such as the ListBox on the View Saved Games page. The font size used for the content inside the ListBox can be adjusted as well as the spacing between each line.

Furthermore, to enhance my application, I could implement sound effects into the game. The user currently already receive constant feedback from the application to understand what's going on through tool tips, the "How To Play" page and visual elements like the turn indicator and bold headings. Nevertheless, sound effects could provide even more insight. For example, I could make it so that whenever a player lands on a ladder or a snake, there would be a sound effect notifying the user.

MAINTAINING THE SOFTWARE SOLUTION

MAINTENANCE CHANGES

It is necessary to regularly upgrade the software solution as requirements change and as errors are found that require correction. The maintenance prodcedures involve continuous modification of the code.

Debugging was done as the code was devleoped. However, there were still changes made after all the necessary features were implemented, recorded on the following logs.

Patch 1

08/07
Tasks Achieved

- Debugged the resume-game function, things from lists were going out of range
- Coded method that deletes the game from storage when it's finished

The resume-game function was debugged to prevent exceptions/errors so the game can run smoothly.

When a game is finished, it will delete the game from the saved games list automatically to prevent the user from resuming a finished game.

Patch 1.1

11/07
Tasks Achieved

- Debugged an exception; now prevents the user from setting a long game name

The user could spam the textbox where they set the game name; if the string was too long, there would be an exception as it would not be able to create a textfile with such a long name. A try-catch was implemented to prevent this from happening.

Patch 1.2

15/07
Tasks Achieved

- Improvements on screen flickers/glitches when a new form is loaded

Whenever a new form loads, the form's window-state will be set to normal in the load method to fix the screen flickers.