Agile Group Project 2018
Design Document

Introduction
This document outlines the design for the Toguz Korgool game. It contains both the class diagram and the use case diagram for playing Toguz Korgool. In addition to this, wireframes for the user interface are also included.

Github
For our project we used github to keep a remote repository of our work. At the start of the project we committed to the master branch, however, we realised this was an error and should not be done and since used separate branches for frontend and backend features, only merging them with master when the features were complete and reviewed by the rest of the team.

Requirements Implemented
<u>Base Requirements</u>
- Players are able to make moves by clicking on one of the holes on their side, iff it contains korgools and is not a Tuz
- The computer will then make a move and all holes will be updated with the correct number of korgools
- Capturing a Tuz works as described in the brief
- Capturing korgools works as described in the brief
- Game ends when player or computer has captured 82 korgools
- Able to create custom games with differing number of korgools in holes and kazan
- Able to save game state

<u>Extended Requirements</u>
- More advanced AI has been implemented. This AI will try to make a move to gain a tuz, otherwise it will try and make a move which will capture the most korgools.
- Interface has been customised. As both the player and AI capture korgools, these are shown as circles stacking inside the kazan.

Requirements Pending Implementation
- Custom game setting should have a feature which allows a hole can be set as a Tuz at the start of the game

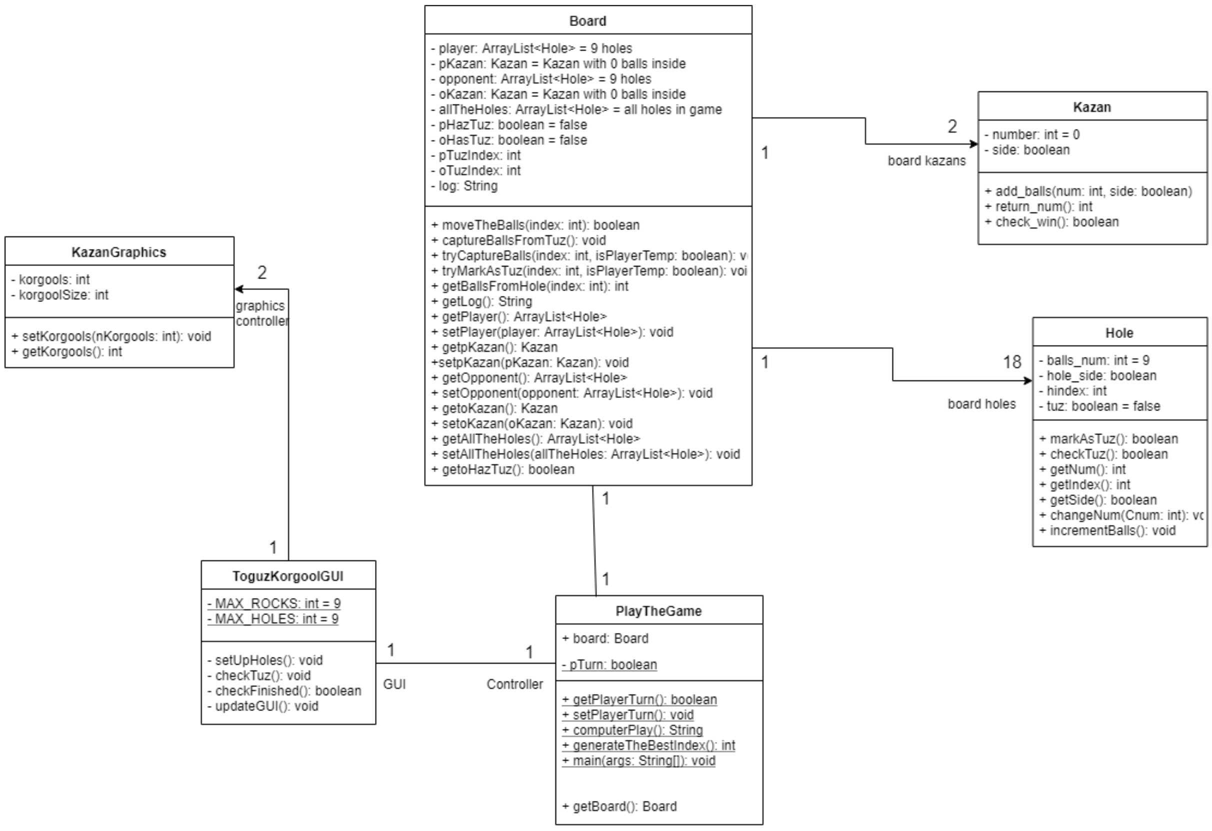
Known Bugs
- Currently there is a bug with the testing where the test will crash where swing seems to be unable to find components when a second GUI test class is run.

Opportunities for Future Development
- Playing against another human player could be implemented by adding a button to the start page, which then opens a window similar to the play vs ai page, but with all hole as active buttons and players could take turns playing against each other.
- User interface could be further improved by changing the colour of buttons and the background to create a more consistent aesthetic to the application.

Overall Architecture

For our project we decided to split the front end (e.g. GUI) and back end (e.g. calculating if a move causes a Tuz) features into separate classes as much as possible. This can be seen in the class diagram below.

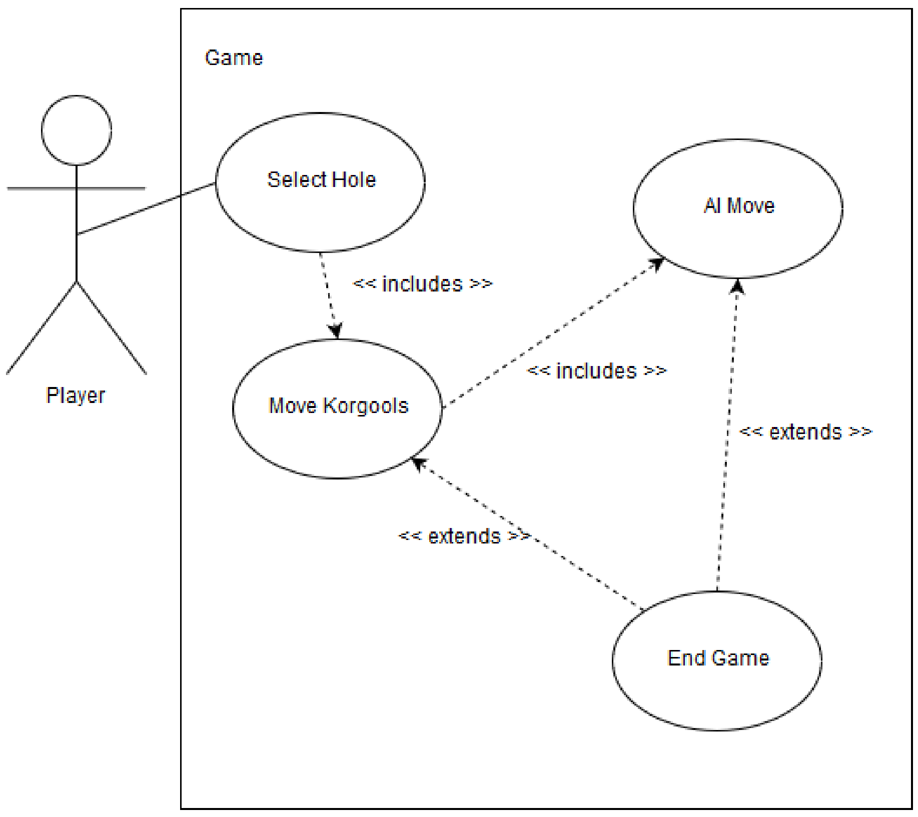
<u>Class Diagram</u>



Board
- player: ArrayList<Hole> = 9 holes
- pKazan: Kazan = Kazan with 0 balls inside
- opponent: ArrayList<Hole> = 9 holes
- oKazan: Kazan = Kazan with 0 balls inside
- allTheHoles: ArrayList<Hole> = all holes in game
- pHazTuz: boolean = false
- oHasTuz: boolean = false
- pTuzIndex: int
- oTuzIndex: int
- log: String
+ moveTheBalls(index: int): boolean
+ captureBallsFromTuz(): void
+ tryCaptureBalls(index: int, isPlayerTemp: boolean): v
+ tryMarkAsTuz(index: int, isPlayerTemp: boolean): voi
+ getBallsFromHole(index: int): int
+ getLog(): String
+ getPlayer(): ArrayList<Hole>
+ setPlayer(player: ArrayList<Hole>): void
+ getpKazan(): Kazan
+setpKazan(pKazan: Kazan): void
+ getOpponent(): ArrayList<Hole>
+ setOpponent(opponent: ArrayList<Hole>): void
+ getoKazan(): Kazan
+ setoKazan(oKazan: Kazan): void
+ getAllTheHoles(): ArrayList<Hole>
+ setAllTheHoles(allTheHoles: ArrayList<Hole>): void
+ getoHazTuz(): boolean
Kazan
- number: int = 0
- side: boolean
+ add_balls(num: int, side: boolean)
+ return_num(): int
+ check_win(): boolean
1
2
board kazans
Hole
- balls_num: int = 9
- hole_side: boolean
- hindex: int
- tuz: boolean = false
+ markAsTuz(): boolean
+ checkTuz(): boolean
+ getNum(): int
+ getIndex(): int
+ getSide(): boolean
+ changeNum(Cnum: int): vc
+ incrementBalls(): void
1
18
board holes
KazanGraphics
- korgools: int
- korgoolSize: int
+ setKorgools(nKorgools: int): void
+ getKorgools(): int
2
graphics
controller
1
ToguzKorgoolGUI
- MAX_ROCKS: int = 9
- MAX_HOLES: int = 9
- setUpHoles(): void
- checkTuz(): void
- checkFinished(): boolean
- updateGUI(): void
1
GUI
1
Controller
1
1
PlayTheGame
+ board: Board
- pTurn: boolean
+ getPlayerTurn(): boolean
+ setPlayerTurn(): void
+ computerPlay(): String
+ generateTheBestIndex(): int
+ main(args: String[]): void
+ getBoard(): Board

Above is the class diagram. This shows the fields and methods of the classes. A '+' denotes the field/method is public and a '-' is to show it is private. Excluded from this diagram is some of the GUI classes for the start page as they are not relevant to the core functionality of the game as so would be irrelevant and clutter the diagram.

Use Case Diagram



This is the use case diagram for the Toguz Korgool game when playing against the computer.