|  |  |
| --- | --- |
| **Course** | BSCH |
| **Stage / Year:** | 3 |
| **Module:** | HGP |
| **Assignment No:** | Project: Building a Working Game of Go |
| **Deadline:** | 17/12/2017 23:55 |
| **Submission:** | Upload to Moodle |
| **Weighting:** | 100% of project (60% of module grade) |
| **Group size:** | 2 |

# 1. Introduction.

**N.B.:**

* **Please read the project description in full before you start coding. Failure to do so will result in you missing out on important information and reducing your ability to avail of marks. Please attempt to work in line with the file structure provided. Very large departures from this template may result in loss of marks.**

**9x9 board use a 7x7 board for your app**

* In this project, you will be tasked with building a fully working game of Go.

**White Stone’s last liberty Stone**

**White Stone**

**White Stone**

**Black Stone**

|  |  |
| --- | --- |
| Figure 1. White can be captured by black placing a stone on position E4. | **White Stone**  **Two Eyes Stone**  Figure 2. This white string is safe as it has 2 eyes. |

# 2. Resources

* **Moodle**
  + This Description
* **From your lab work** 
  + Example20. The solution is heavily based on this and it is essential to have a detailed working knowledge of Example20 before proceeding with this project.
  + Reversi. The skills you learned in the design of your Reversi application will stand to you. You may like to begin with a working version of Reversi.

# 3. Explanation of the Game

Go ("encircling game") is an abstract strategy board game for two players, in which the aim is to surround more territory than the opponent.

The game was invented in China over 3’000 years ago, and is therefore believed to be the oldest board game continuously played today. It was considered one of the four essential arts of the cultured  aristocratic Chinese scholars in antiquity. Despite its relatively simple rules, Go is very complex, even more so than chess.

Computers have only recently been capable of beating human masters. Have a look at the following for more details: <https://deepmind.com/research/alphago/>.

## Initial board layout:

We will use a 7x7 board to ensure quick game play and reduced complexity. Go is commonly played on a 13x13 and 19x19 grid. Black goes first. Stones are placed on the grid intersections.

Movement:

Black plays first, with black and white taking turns. A *stone* can be placed at any unoccupied intersection of the board with limited exceptions.

1. Suicide rule: You cannot place a stone which will immediately have no liberties. [https://youtu.be/JWdgqV-8yVg?t=9m00s](https://youtu.be/JWdgqV-8yVg?t=7m35s)
2. KO Rule (Eternity Rule) Previous game states are not allowed. Keep a list of previous game states which must be checked before stones are placed <https://youtu.be/JWdgqV-8yVg?t=7m35s>

## Determining a Winner

When a player thinks their territories are all safe, they can't gain any more territory, reduce their opponent's territory or capture more strings, instead of playing a stone on the board they pass and hand a stone to your opponent as a prisoner. Two consecutive passes ends the game.

Points are awarded for

* stones captured
* territory controlled by a colour

## Additional Information

* A detailed set of rules is available here <https://www.britgo.org/intro/intro2.html>.
* A cartoon tutorial is available at <https://www.britgo.org/cartoons/index.html> .
* A well-structured version of the rules is available here <https://en.wikipedia.org/wiki/Rules_of_Go>
* An extensive list of GO terms available at <https://en.wikipedia.org/wiki/List_of_Go_terms> . You are not required to know these to complete the project but they will expand your awareness of the game.
* There is a lot of additional information on Go some of it code related at <https://senseis.xmp.net/>

## Interesting situations

* Seki (Impasse): A board position may arise where a play can capture opponents piece but the oppoenent can immediately recapture a string of pieces. Whoever goes first loses. <https://youtu.be/JWdgqV-8yVg?t=11m12s>
* Having *eyes* is a strong position

## Handicaps

You may implement this as an advanced task but you will need to do more research. Typically, white gets 7.5 points for going 2nd. The .5 point is to avoid a tie.

# 4. Comparison between Go and Xs & Os

* Below is a table showing some of the files that you MUST have in your solution and their counterparts in Example020 with comparisons

|  |  |  |  |
| --- | --- | --- | --- |
| **File name in Project** | **Draughts** | **File name in  Example20** | **Xs & Os** |
| **Go.java** | * Has a custom control with a GoBoard. | **Example020.java** | * Has a custom control with an Xx & Os board. |
| **GoBoard.java** | * 7x7 square grid. * Grid division is highlighted with horizontal and vertical lines * Pieces can be placed at empty intersections. * Maps mouse clicks to intersections. * Valid moves are defined in GameLogic.java. | **XOBoard.java** | * 3 x 3 square grid. * Grid division is highlighted with horizontal and vertical lines * Pieces are placed within empty grid squares. * Maps mouse click grid squares. * Valid moves were determined in this class. |
| **Stone.java** | * 2 distinct stones – (white and black) | **XOPiece.java** | * 2 distinct pieces (X and O) |
| **CustomControl.java** | * Detects mouse clicks, and keyboard actions | **CustomControl.java** | * Detects mouse clicks and keyboard actions |
| **CustomControlSkin.java** | * Identical | **CustomControlSkin.java** | * Identical |
| **GameLogic.java** | * Determines if a move is valid. * Determines when a winner has occurred * Additional methods can feature here too. |  | * Not present |
| **Additional Classes** | * To be added with justification |  | * Not present |

# 5. Deliverables

* + The full list of deliverables is

1. This document with the following sections completed
   1. Summary of Division of Work Word
   2. Screen Shots of Working Application
2. Folder containing all the java source files, images etc. relevant to your application
   * You are **both** required to submit a single **identical** archive (zip) file to Moodle that has a filename of **<lastname1>\_<firstname1>\_<studentnumber1>\_<lastname2>\_<firstname2>\_<studentnumber2>\_go.zip** If the files are not identical (MD5 hash etc.) then you will be called to interview.
   * In that zip file, I should find a single folder containing all your deliverables and it should be named **<lastname1>\_<firstname1>\_<studentnumber1>\_<lastname2>\_<firstname2>\_<studentnumber2>\_go**

N.B. work is expected to BE YOUR OWN OR YOUR PARTNERS. **PLAGIARISM** will not be tolerated under any circumstances.

# 6. Tasks & Marking

Below is a list of tasks which must be completed and their associated marks

|  |  |  |
| --- | --- | --- |
| Task Number | Description | % of Grade |
| 1 | Generate an application that is forced to be square in size. It should display a full Go board of side size 7. | 10 |
| 2 | Add code and menus/buttons/labels to your application to   1. How to play your game including rules 2. Show how many prisoners each player has taken 3. Show how much territory is controlled by a player 4. Show whose turn it is 5. Allow player to pass 6. Allow the game to be reset | 20 (6 x 3.3) |
| 3 | Implement placement of stones using mouse clicks. | 10 |
| 4 | Implement placement of stones in valid locations only – suicide rule | 10 |
| 5 | Implement placement of stones in valid locations only – KO rule | 10 |
| 6 | Implement capture of stones – single stone | 10 |
| 7 | Implement capture of stones – multiple stone | 10 |
| 8 | Implement winner detection, the game should then end immediately with an appropriate notification. Two passes. | 10 |
| 9 | Additional features select one of the following:   * 2 timers 1 for each player to implement speed Go. Each player should have 2 minutes to make moves. The 1st players timer should start to count down when the game is started, 2nd players timer counts down when 1st player has completed his move and so on. If a player runs out of time then they will lose the game. * Animation of moves * Implement a handicapping system * Other additional feature of your choice with similar complexity | 10 |
| Please Note that Marks will be Deducted for the following:   * Code that fails to compile -30% * Submission in wrong format. Your project should be submitted in zip format only -10% * Submission with the wrong filename -10% | | |

# 7. Working in Pairs

* Agree on a common IDE for code development. Converting from one project format to the other is messy at best.
* If you can use a SCM (source code manager) to develop and share code please do. I would recommend using GitHub if you can (particularly if you have access to a git server somewhere where all changes can be uploaded).
* Decide on a development strategy. Try to work on independent parts if you can. Working in parallel will cut the time in half.
* “Weeks of programming can save you hours of planning” a quote from Scott Meyers. Basically, it means design the project first before implementing it. This means deciding on your data structures, UI design, view design etc. before you start. Pay attention to your data structure for the game board and how you implement the rules.

# 8. Suggested Development Plan

Steps to complete this project

**Day 1 - Learn the Game**

1. View “How to play go in 2 mins” <https://www.youtube.com/watch?v=Jq5SObMdV3o>
2. View “How to play go in 15 mins” <https://www.youtube.com/watch?v=JWdgqV-8yVg>
3. Read the rules of Go <https://www.britgo.org/intro/intro2.html>
4. Read the online cartoon tutorial <https://www.britgo.org/cartoons/index.html>
5. Play a game of go online <https://www.cosumi.net/en/>
6. Play go with your partner <http://ba.net/juegos/go/#blank-9>
7. Discuss your understanding of Go with your team.
8. Discuss your understanding of Go with another team.

**Week 1 - Hack Your Code – task 1, 2, 3**

1. Determine which partner has a better working version of Reversi and use this as the base for you project. You could also use Example20.
2. Share this using GitHub (preferred) or google drive etc.
3. Start the *Division of Work Log* deciding how the work is to be divided between partners – a google drive document would be good for this.
4. Task1 One member could work on adapting the appearance of the Reversi GUI to mimic GO while the other researches the game logic that will be required to implement GO.
5. Task 2 - Add in the controls. You can connect them up later.
6. Task 3 – Implement the placement of stones.
7. Discuss your progress with your team mate and other teams.

N.B. Make sure to test your application after each small change.

Outcome: At the end of this phase your GUI should LOOK LIKE a game but most of the functionality will not be present. You should be able to place stones on the board.

**Week 2 - Workout Game Logic - Design for task 4,5,6,7,8**

This is the most complex part of your project. You should have a good idea on paper about how this game should be played before you implement this in GameLogic.java.

1. Discuss game logic with your group
2. Discuss game logic with other groups
3. Agree on what methods should be written, what they will do and what they will return.
4. Compare these methods on a logical level with other teams so you know you are on the right track. DO NOT SHARE CODE with other groups.

Warning If you have spent time working out the game logic on paper/in written documentation then the next part will run more smoothly. If you attempt to hack your code to get it working you will fail in the next step. You will not accidently write the correct code … planning is essential.

**Week 3 – Implement the Game Logic – Implementation for task 4,5,6,7,8**

1. For each of the methods that you have defined generate detailed comments in GameLogic.java prior to writing the methods.
2. Define tests which you can run to determine if the code is performing correctly. This testing should be detailed and performed often. When you have a small code base errors are easy to find. As your codebase becomes larger it becomes more and more difficult to locate and correct errors. Below are 2 examples of how you can could test:
   1. Formal Test Classes if you have done this previously
   2. Text output to the console or visual feedback from you GUI.

Outcome: You should have a game that can be played but contains a number of errors (large or small) you should have some code in place which is helping you figure out where specifically the errors are located.

**Week 4 – Debug Code**

You will be focusing on generating a working game. You will have a number of errors and you will be figuring out if they are logical errors (i.e. you made a mistake in the work in Week 2) or if they are implementation bugs (i.e. you made an error in Week 3).

**Week 5 – User Acceptance Testing & Advanced Features**

1. Ask other students to test your code to see which features are present and which are not.
2. Work on adding the advanced features. Ensure that it is easy to roll back the code to previous states.

# 9. Division of Work Log

Student Name1: Claudio Dantas Student Number1: 2953941

Student Name2: Gerard Cunningham Student Number2: 1234567

Please complete the sections below with regard to the estimate of the division of work between the two partners

If the work was split in the range of 45% to 55% per partner then that is fine and simply say “Work was evenly divided”. If this was not the case then state with a summary sentence. This is the important statement of this file.

Division of work: work was evenly divided \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Code repository log (if applicable)

## <https://github.com/DantasClaudio/GoProject/commits/master>

## Percentage of work completed by each partner on each class / task

Some areas require more work than others so this is only for reference. An average of these values will not be calculated.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Filename / Task** | **Claudio Dantas** | **Notes** | **Gerard Cunningham** | **Notes** |
| GoBoard.java | 100% |  | 0% |  |
| Stone.java | 100% |  | 0% |  |
| Go.java | 100% |  | 0% |  |
| CustomControl.java | 100% |  | 0% |  |
| CustomControlSkin.java | 100% |  | 0% |  |
| Learning Rules | 50% |  | 50% |  |
| GitHub Repository | 100% |  | 0% |  |

# 10. Screen Shots of Working Application

**N.B. Be sure to comment on what is working and not working for each of the tasks. The boxes should be expanded to contain images and description.**

|  |
| --- |
| **Task 1 - 2 images in total (2 image of board with differing width + description of what is working/not working)** |
| Everything worked properly from the beginning |

|  |
| --- |
| **Task 2 – 3 images in total (1 image of task 2a, 2 images showing the remainder of the subtasks + description of what is working/not working)** |
| Every functionality presented in the photos are working. The player can pass his turn, reset the game and open a window with the basic game rules. |

|  |
| --- |
| **Task 3 – 2 images in total (2 images of the placement of stones + description of what is working/not working)** |
| Placement of stones works as it should be. |

|  |
| --- |
| **Task 4 – 1 image in total (show rule in operation + description of what is working/not working)** |
| Additional rules are not working. |

|  |
| --- |
| **Task 5 – 1 image in total (show rule in operation + description of what is working/not working)** |
| Suicide rule in progress. |

|  |
| --- |
| **Task 6 – 2 images in total (2 images of the capture of single stone, 1 before and 1 after + description of what is working/not working)** |
| White player can capture a Black piece.    Piece was captured by White Player. |

|  |
| --- |
| **Task 7 – 2 images in total (2 images of the capture of multiple stones, 1 before and 1 after + description of what is working/not working)** |
|  |

|  |
| --- |
| **Task 8 (2 images of winner detection + description of what is working/not working)** |
| After each player pass their turn, the game is over. However, winner detection is not working. |

|  |
| --- |
| **Task 9 (3 images of additional feature + description of what is working/not working)** |
|  |