Class Project: Checkers

Gionata Bonazzi

Cole Sellers

Rosa Fleming

Brendan Cronan

CIS 350: Winter 2018

Table of Contents

Project Information

Final Deliverable:

For the final version of our project, we decided to focus on checkers and implementing more features such as different game modes, and timers. In addition, we created an Artificial Intelligence bot, so that users can play against the computer.

First Deliverable:

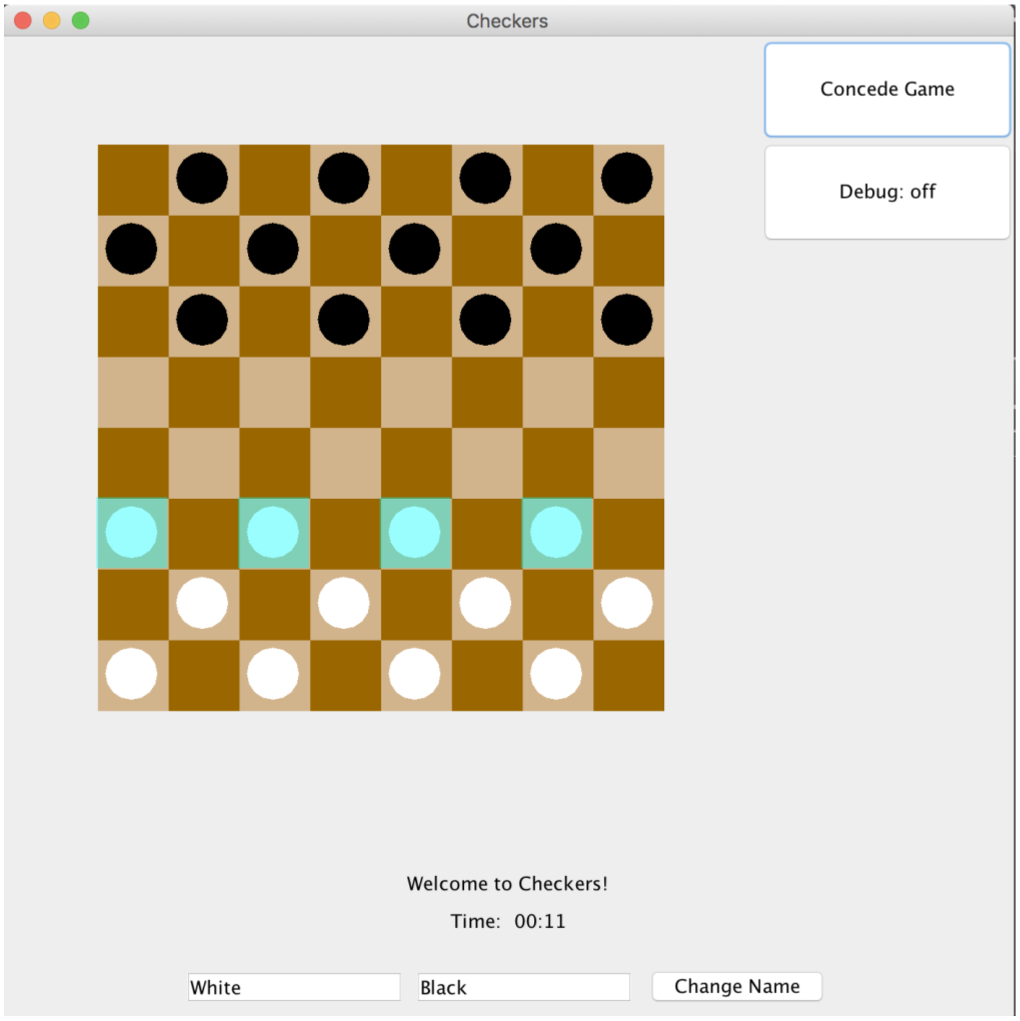
The project is about a suite of board games: the similarity between the games is that they are all played in some kind of grid board.

Final Features:

Our game has many different features that contribute to the overall game. When the game first starts, the user is playing in a “free play mode” against an AI bot. This mode means that the game and the turns are not timed in any way. The other modes the user can play include, a timed game mode and a timed turn mode. The timed mode will allow users to play the game for a set amount of time before ending. The person whose turn it currently is when the timer hits will lose the game. The timed turn mode is when each turn has only a set amount of time to make a move before they lose. These times are represented by a timer at the bottom of the board. These three modes can also be played using a player vs player mode where the AI is not implemented and two users can compete. Another feature we have is that users can change their name in the game. In addition to the two modes, AI and player v player, the user can turn on a debug mode to see in the console a list of what moves were made. At any point if the user wants to quit a game or return to a previous menu screen you can click the back button or concede button. Lastly, the main features of the game are that the checker pieces that can move or have to move will be highlighted and then once they are clicked the spots they can move to when then be highlighted. The game will follow standard checkers rules.

Screenshots

Final Deliverable:



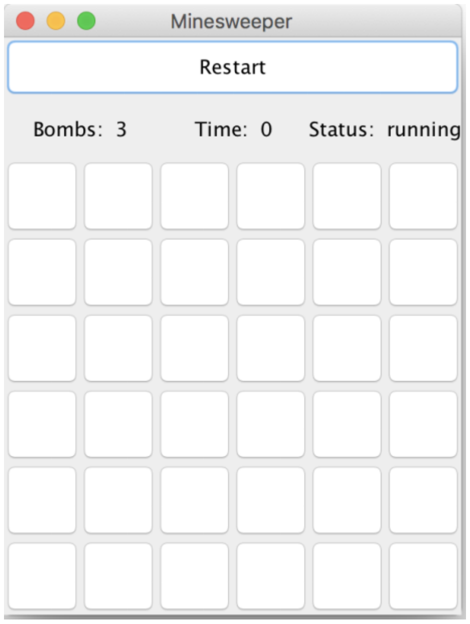
Welcome to Checkers!

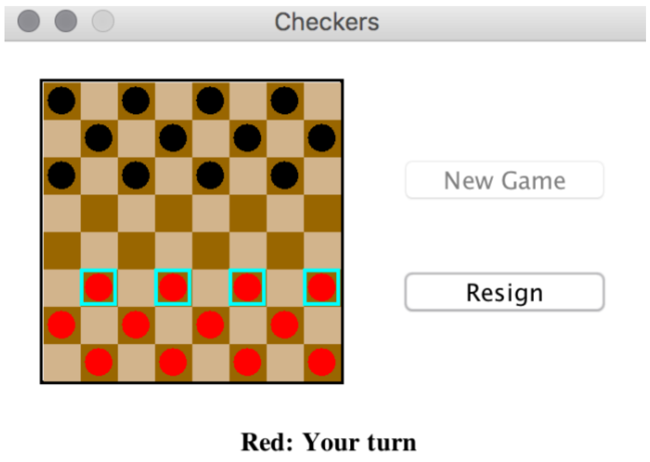
Time: 00:11

White

Black

First Deliverable:





Project Plan

Final Deliverable:

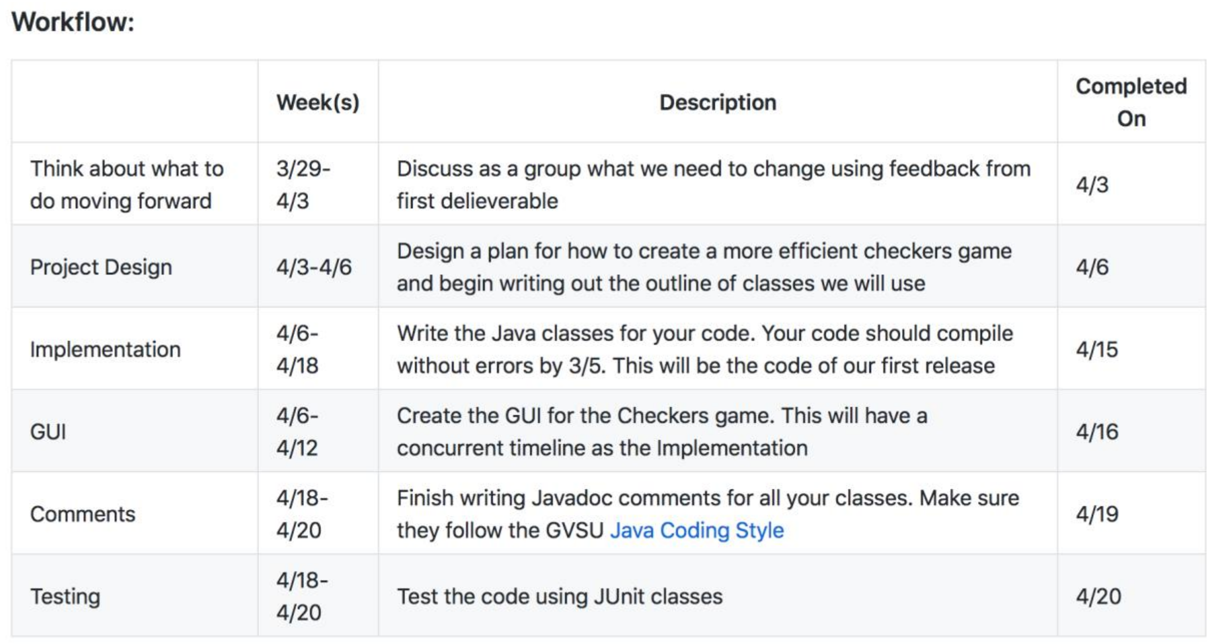
For the second release we decided if we continue on the same path we would never have enough time to implement all of the features we wanted. We cut it down to just checkers and we converted to a waterfall methodology where we used our debug mode to get the code working then did all of our test cases after the program was functional. We implemented an AI as well as many different game modes within the checkers game to make the project worthy. We also implemented a debug mode which helped fix errors when writing the initial programs. From our first release to our final we rewrote the entire checkers program to make it Object Oriented and very efficient.

First Deliverable:

For the overall project we decided to follow an incremental phased development. For the first deliverable only, we decided to follow the Agile model to deliver a simple, but functioning, program. We decided to split the project in two parts: parts to be ready for the first deliverable (Checkers, Minesweeper), because they were the fastest to prepare, while working on Sudoku and a GUI to be ready before the final release. The team one, responsible for the first delivery, is made up by Rosa and Cole, while team two is made up of Gionata and Brendan. Since team two has more time to work on their code, for the first deliverable they also made up the umbrella activities' team. Unfortunately, our schedule didn't always match so we based our communication on group messaging through Group Me, and the delay of such communication method forced us to choose to code the two games in the first deliverable independently. In the end, the first deliverable is composed of two independent projects: Checkers, and Minesweeper.

Table of Features and Risks (Replaces Gantt Chart)

Final Deliverable:

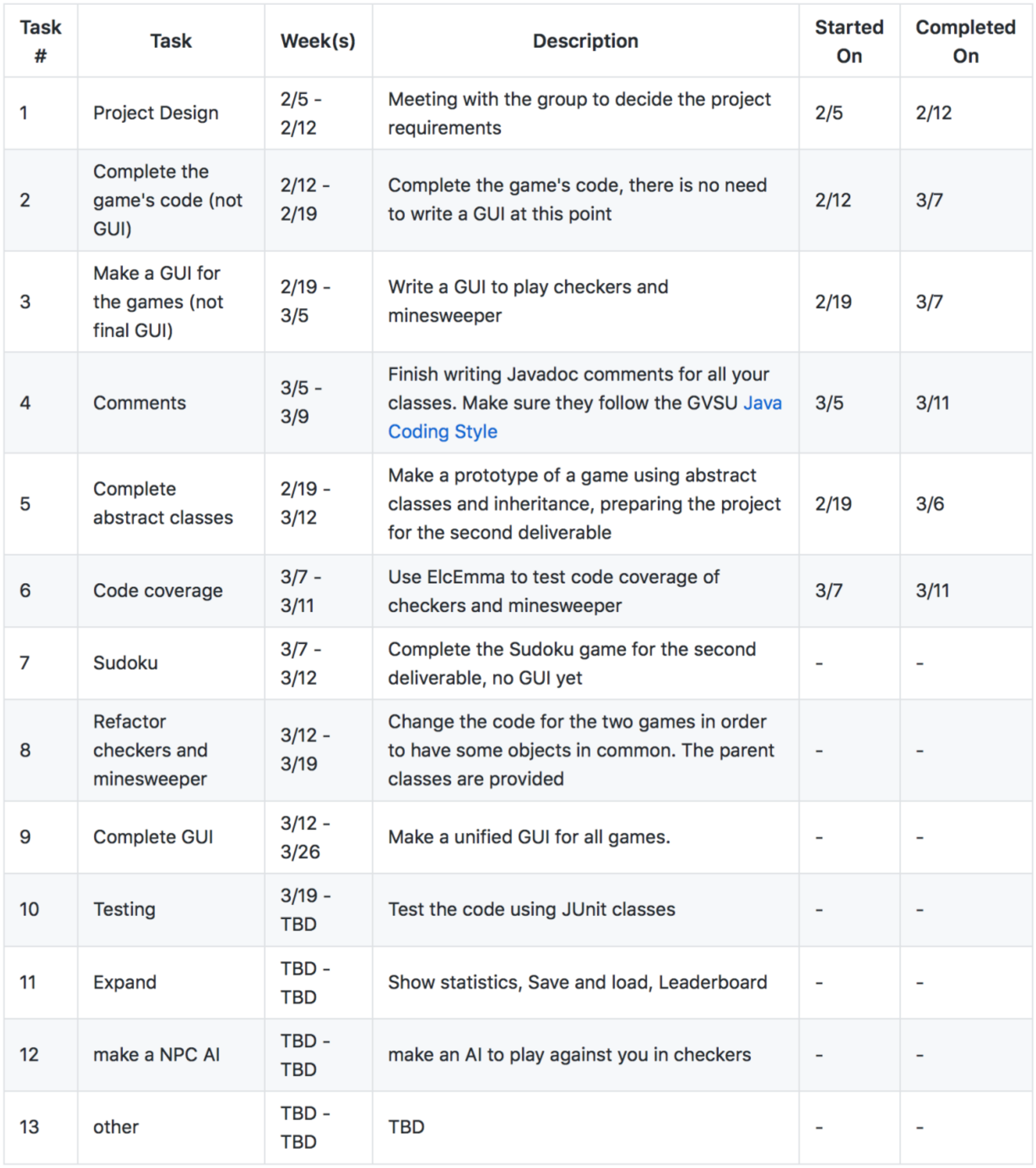


Workflow:

	Week(s)	Description	Completed On
Think about what to do moving forward	3/29-4/3	Discuss as a group what we need to change using feedback from first delieverable	4/3
Project Design	4/3-4/6	Design a plan for how to create a more efficient checkers game and begin writing out the outline of classes we will use	4/6
Implementation	4/6-4/18	Write the Java classes for your code. Your code should compile without errors by 3/5. This will be the code of our first release	4/15
GUI	4/6-4/12	Create the GUI for the Checkers game. This will have a concurrent timeline as the Implementation	4/16
Comments	4/18-4/20	Finish writing Javadoc comments for all your classes. Make sure they follow the GVSU Java Coding Style	4/19
Testing	4/18-4/20	Test the code using JUnit classes	4/20

We replaced the Gantt chart with the workflow table, we found this to be more practical when it came to keeping up to date with production.

First Deliverable:



Task #	Task	Week(s)	Description	Started On	Completed On
1	Project Design	2/5 - 2/12	Meeting with the group to decide the project requirements	2/5	2/12
2	Complete the game's code (not GUI)	2/12 - 2/19	Complete the game's code, there is no need to write a GUI at this point	2/12	3/7
3	Make a GUI for the games (not final GUI)	2/19 - 3/5	Write a GUI to play checkers and minesweeper	2/19	3/7
4	Comments	3/5 - 3/9	Finish writing Javadoc comments for all your classes. Make sure they follow the GVSU Java Coding Style	3/5	3/11
5	Complete abstract classes	2/19 - 3/12	Make a prototype of a game using abstract classes and inheritance, preparing the project for the second deliverable	2/19	3/6
6	Code coverage	3/7 - 3/11	Use ElcEmma to test code coverage of checkers and minesweeper	3/7	3/11
7	Sudoku	3/7 - 3/12	Complete the Sudoku game for the second deliverable, no GUI yet	-	-
8	Refactor checkers and minesweeper	3/12 - 3/19	Change the code for the two games in order to have some objects in common. The parent classes are provided	-	-
9	Complete GUI	3/12 - 3/26	Make a unified GUI for all games.	-	-
10	Testing	3/19 - TBD	Test the code using JUnit classes	-	-
11	Expand	TBD - TBD	Show statistics, Save and load, Leaderboard	-	-
12	make a NPC AI	TBD - TBD	make an AI to play against you in checkers	-	-
13	other	TBD - TBD	TBD	-	-

While not being up to date anymore, the workflow for the first deliverable is a good record of our struggles as a group and the change we made from the first deliverable to the last one.

Requirements & Definition

Final Deliverable:

For the final deliverance we decided to make a efficient game of checkers, we dropped the initial suite of board games due to time constraints we decided it would be better to create a great game of checkers vs 4 games that were subpar. We have one main GUI that runs the game and has buttons to change each game mode or player vs player and player vs AI game modes. We split the game up into multiple java classes to help with testing and setting the game up so if in the future more games were to be added it would be possible.

First Deliverable:

For the first deliverable we are going to have two games: Minesweeper and Checkers.
As of the first deliverable, these two games are separate projects and are played as stand-alone games.
Each game has a different, but simple, GUI and does not keep any information or statistics about the game.
For the second deliverable, these two games, along with Sudoku, will be put in a single project folder and run through a unified GUI. To make testing and maintenance easier, each game will have objects in common, and the code will be recycled as much as possible.

Development

Final Deliverable:

Contrary to our first deliverable, for our second portion we realized a lot of our time constraint issues were caused by coding style differences and we could work more efficiently at the end if we worked as a group. So, to finish out our project, we worked on a large screen and all worked on the same code. This was how we finished out the rest of the project.

First Deliverable:

To make the project go smoother, we as a team decided to assign one game to each team member, and the GUI to the member left.
For that reason, the team members worked independently on each game.
Rosa's Minesweeper and Cole's Checkers currently do not have anything in common as of now.
Gionata is working in writing classes to make full use of encapsulation and polymorphism; for that reason, Sudoku will not be in the first deliverable, since its

current form is a prototype of what the final deliverable will look like and it’s not completed.
Brendan is working on a unified GUI using the objects and classes made by Gionata, and we decided to push the deadline for the GUI to the second deliverable.

Verification

Final Deliverable:

For the second release since we decided to change to the waterfall method we had a clear-cut picture in what we wanted the final game to look like. Therefore, we had all of the features we wanted completed before testing.

First Deliverable:

For the first deliverable, the project looks different from the project requirements we decided. Since we are following an incremental process method, we decided to settle for a simpler version of the project for the first deliverable, one where only the basic elements of each game work.

Maintenance

Final Deliverable:

After analyzing checkers and deciding to write an AI bot for it, we chose to redesign our game to increase efficiency. Maintenance will be easy for this program, as well as the included Debug mode which will help easily find the bugs that happen to arise and tells us where the bug has occurred.

First Deliverable:

The code in the first deliverable was not created to be maintained for long. Instead some structures we used will be moved in more generic Objects that will be implemented by every game. For that reason, the first deliverable has not been tested using JUnits or FindBugs, because we figured testing code that we know won’t be passed over in its entirety is a waste of resources. Instead, we focused on having some simple working games made by the first team, while the second team smothered the design and requirements for the final version of the project. For that reason, team two will help team one to transition their old code into the new design.

Umbrella Activities

Final Deliverable:

Meeting Date	Task	Hours
3/29	Discuss first deliverable critiques and create a plan for moving forward	2
4/3	Design a plan for how to create a more efficient checkers game and begin writing out the outline of classes we will use	2
4/6	Redeveloped the skeleton of the board	1.5
4/12	Redeveloped the main logic of checker game to be more efficient	2.5
4/13	Created AI bot	2
4/14	Debugged checkers main logic	4.5
4/15	Debugged checkers main logic, got it to finally run correctly for every move type	7
4/16	Added more buttons and features while creating the presentation, also worked on code coverage	4
4/18	Worked on final documentation and testing, including checkstyle and junit	3.5

First Deliverable:

These activities have been covered for the most part by team two.
Part of the umbrella activities are meeting documentation through a markdown file in the GitHub repository and available to everyone, and weekly checks on the repository and branches to see if everything is working correctly, and if the team is keeping with the overall organization: folders, packages, and general nomenclature of files. Another important umbrella activity pertained the schedule: from its creation, to its maintaining, and following up with the other people to see if everyone is following the schedule and its deadlines.

First meeting: Discussion about the project’s requirements. After deciding the requirements, we came up with a schedule, and divided the work between the team members. We divided the work up equally to ensure everyone knew what part they would be working on, trying to give everyone a part that they were comfortable with.

Second meeting: An overall look on what to expect for the second deliverable, especially how the final project would look like. We also discussed about potential problems with GitHub, tried to set up the environment on our computers, and talked about the GUI.
Third meeting: Started working on the report, made up a division of work for the report, played the two games in the first deliverable to see if they work. Finally, discussing about future meetings.

Responsibilities

Final Deliverable:

Although we had originally separated our project by games and each person would write their own game and one person would combine all of these into a GUI. After the first deliverable we realized quickly that our games were not compatible and decided to focus on one game. We then did not split up the game by responsibilities but instead split up the project by deadlines and completed each task together. While we were meeting if some people were coding a certain part of the project and did not need assistance other members were assigned to begin designing other features or helping to create a better AI strategy.

First Deliverable:

- Rosa: will create the minesweeper game and will create a GUI for the user to play Minesweeper, as well as its documentation to be included in the report.
- Gionata: is creating the abstract classes and objects to be implemented in each game to be used in the second deliverable, as well as keeping notes on the meetings and uploading them on the GitHub.
- Cole: will create the checkers game and will create a GUI for the user to play checkers, as well as any documentation about checkers used in the report.
- Brendan: will begin to write abstract classes and objects to start the main GUI. Brendan’s responsibility is mostly to keep in contact with everyone to understand better what everyone wants in a future GUI.

Requirements & Specification

Final Deliverable:

Since we changed into a waterfall methodology we had end goal requirements, listed under the high level requirements below. After those goals were met, we wrote Junit tests to test the program.

First Deliverable:

Due using an incremental approach, we decided to have two set of requirements: requirements for the first deliverable, and requirements for the final deliverable. This report includes only the requirements for the first deliverable and will be expanded with the final requirements before the final release is due.

High Level Requirements

Final Deliverable:

For the final deliverable we had many different requirements for checkers.

- 3 game modes:
  - Free Play - There is no time limit.
  - Timed Games - Each game has a certain amount time.
  - Timed Turns - Each turn has a timer, if time runs out you lose
- 2 ways to play:
  - Player vs Player
  - Player vs AI
- Debug Mode:
  - Mode to help detect any bugs while playing.

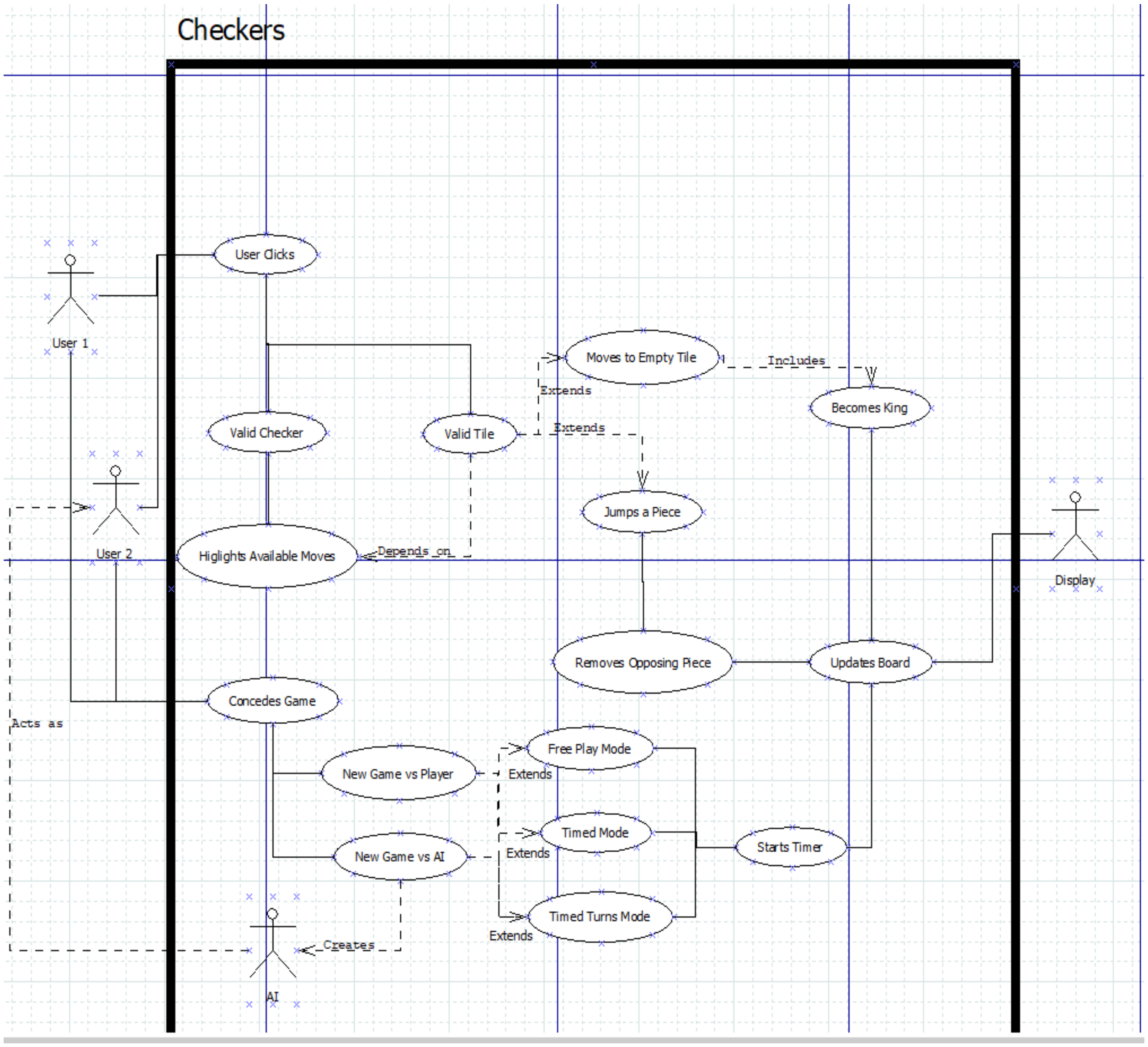
First Deliverable:

The requirements for the first deliverable are simple:

- Have two working games, minesweeper and checkers;
- The user should be able to play the game from start to finish;
- Checkers will be a two players game, it won’t be played against the computer;
- Each game will have a simple GUI;
- Both games must follow the standard rules of each game.

Use-Cases

Final Deliverable:



Use Case Descriptions:

Use Case Descriptions:

Use case: User clicks
Actors: User 1, User 2
Description: The user clicks on a piece which then selects the piece to be moved.

Use case: Valid Checker
Actors: User 1, User 2
Description: Returns if the checker you chose is valid or not. To be valid, a checker has to be able to either jump or move in that turn. Selecting the same checker twice will “deselect” it.

Cross Ref.: The user has yet to select a checker, or the user selected a valid checker and wants to select a new one.

Use case: Highlights available move
Actors: User 1, User 2, Display
Description: Highlights the available moves for the selected piece.
Use-Case: Valid Checker.
Cross-Ref.: The user selected a valid checker but pressed on it only once.

Use case: Concedes game
Actors: User 1, User 2, Display
Description: The user who selected to concede the game loses to the other player. The winner will be displayer and the game stopped; buttons to start a new game will appear. If a User loses the game by the rules, s/he/it "concedes" the game.
Cross-Ref.: The user clicked on the "Concede" button.

Use case: New Game vs AI
Actors: User 1, User 2, Display
Description: The display will show a button for each game mode, and then will start a new game against the computer with the selected mode.
Use-Case: Concedes game.
Cross-Ref.: The previous game has to be over.

Use case: New Game vs Player
Actors: User1, User 2, Display
Description: The display will show a button for each game mode, and then will start a new game against another user with the selected mode.
Use-Case: Concedes game.
Cross-Ref.: The previous game has to be over.

Use case: Valid Tile
Actors: User 1, User 2.
Description: After selecting a valid checker, it will return the valid tiles that the checker can be moved or jumped to.
Use-Case: Valid Checker.
Cross-Ref.: The user selected a checker.

Use case: Moves to empty Tile
Actors: User 1, User 2.
Description: If the piece is moved and the move is legal the piece is then moved to that tile.
Use-Case: Valid Tile.
Cross-Ref.: The user selected a valid tile.

Use case: Becomes King

Actors: User 1, User 2, Display.
Description: if any piece reaches the end of the board it is made a king, and the display updates the board putting a “K” on the checker king.
Use-Case: Moves to empty tile or Jumps piece.
Cross-Ref.: The checker must be on the opposite side of the board to become a king.

Use case: Jumps piece
Actors: User 1, User 2
Description: If an opposing piece is able to be jumped then piece is jumped.
Use-Case: Valid Tile.
Cross-Ref.: A checker has to be close to an enemy checker, followed by an empty space.

Use case: Removes opposing piece
Actors: None
Description: the game removes the piece that has been jumped.
Use-Case: Jumps piece

Use case: Updates Board
Actors: GUI
Description: Updates the board based on the move the user made.
Use-Case: Moves to empty tile or Jumps piece.
Cross-Ref.: A move has been made.

Use case: Starts Timer
Actors: Display
Description: A timer starts as soon as a new game begins. The time limit is defined by the current game mode.
Cross-Ref.: The game is running.

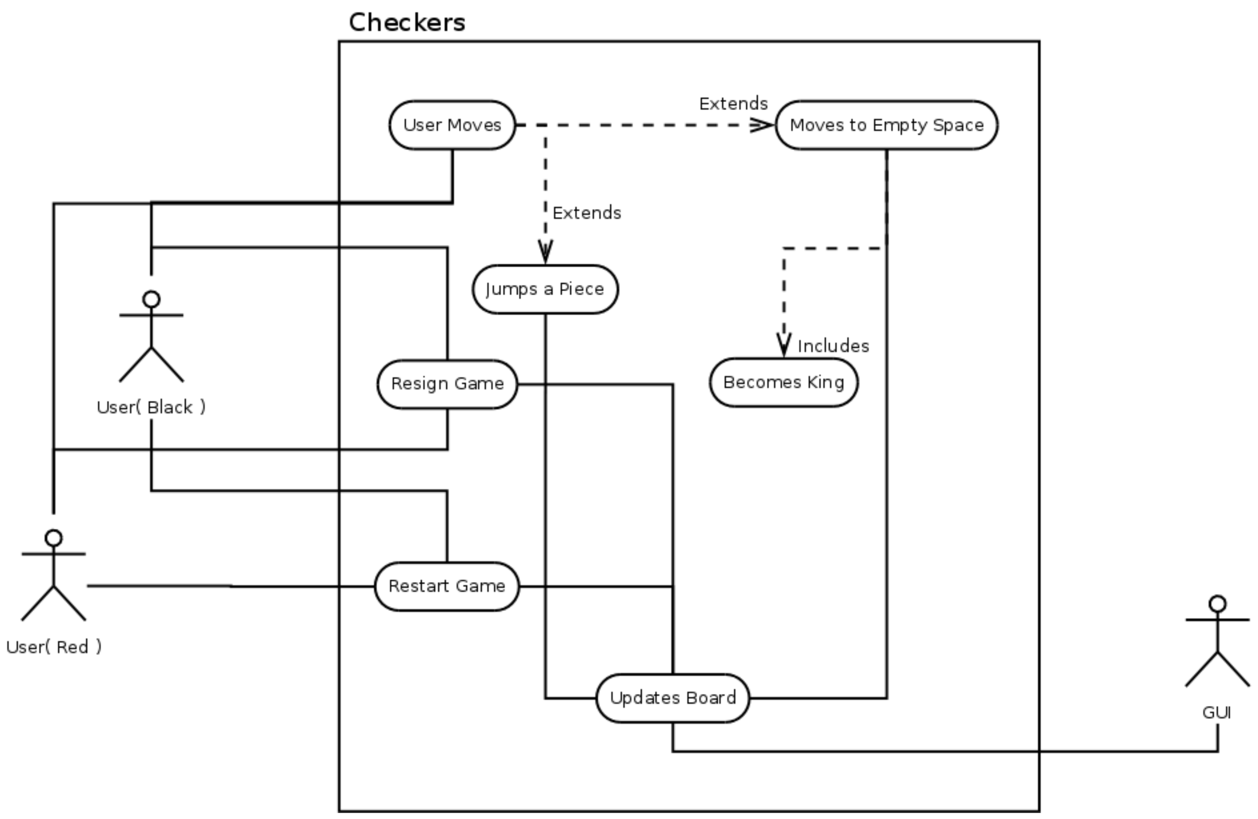
Use case: Free play mode
Actors: user 1, user 2, Display
Description: Sets the game mode to free play in which the user has no set time limit.
Cross-Ref.: The previous game is over, or the software has just opened.

Use case: Timed Mode
Actors: user 1, user 2, Display
Description: Sets the game mode to timed game mode where the user has a specific amount of time per game.
Cross-Ref.: The previous game is over.

Use case: Timed Turn mode
Actors: user 1, user 2, Display
Description: Sets the game mode to timed turn, in which a user has a specific amount of time to complete the turn.

Cross-Ref.: The previous game is over.

First Deliverable:



Use Case Descriptions:

Checkers

Use Case Description:

Use case: User moves
Actors: User(Black), User(Red)
Description: the user will select a piece and an empty square, that piece will then move to the selected space chosen by the user.
Cross ref.: The selected empty square must be a valid space for the selected piece.

Use case: Moves into empty spot
Actors: User
Description: This is extended from the user moves use case, this is where the piece will move when selected to go to a spot.

Cross ref.: The spot must be empty.

Use case: Jumps a piece
Actors: User
Description: This method extends the User moves use case, this is used when a user’s piece jumps a piece to eliminate it from the game.
Cross ref.: There must be an enemy’s piece between the piece and its chosen destination.

Use case: Becomes a King
Actors: User
Description: This is a specific case used by the moves into empty spot use case, the piece becomes a King.
Use-Case: User moves and its extension, Moves into empty spot.
Cross ref.: the selected piece reaches the opposite edge of the board, the enemy’s side.

Use case: Updates board
Actors: GUI
Description: The program updates the screen.
Use-Case: User moves and its extensions.
Cross ref.: The user must have made a move.

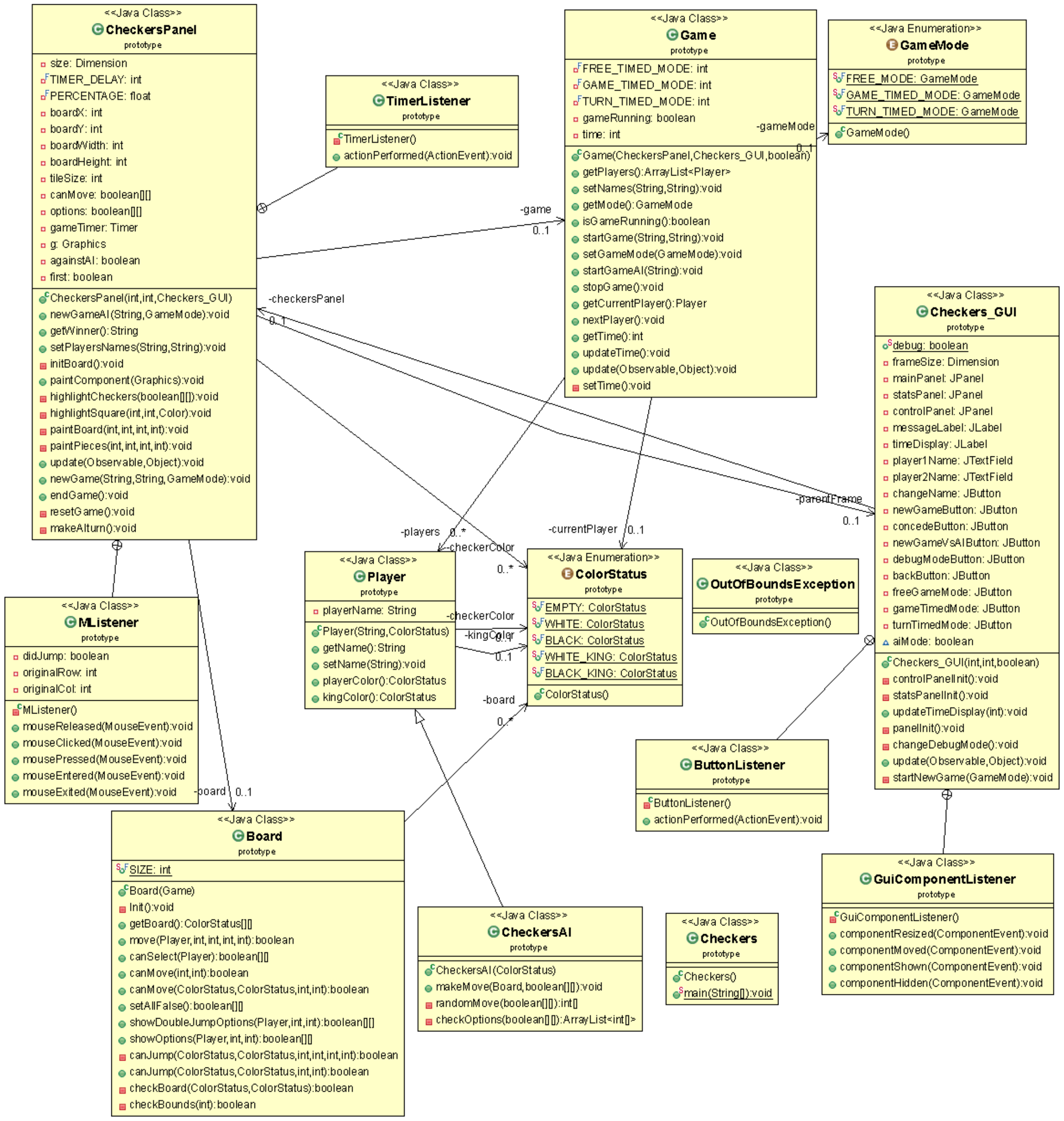
Use case: Resign game
Actors: User(Red), User(Black)
Description: One of the two users resign the game.

Use case: Restart game
Actors: User(Red), User(Black)
Description: One of the two users restart the game.

Design

Final Deliverable:

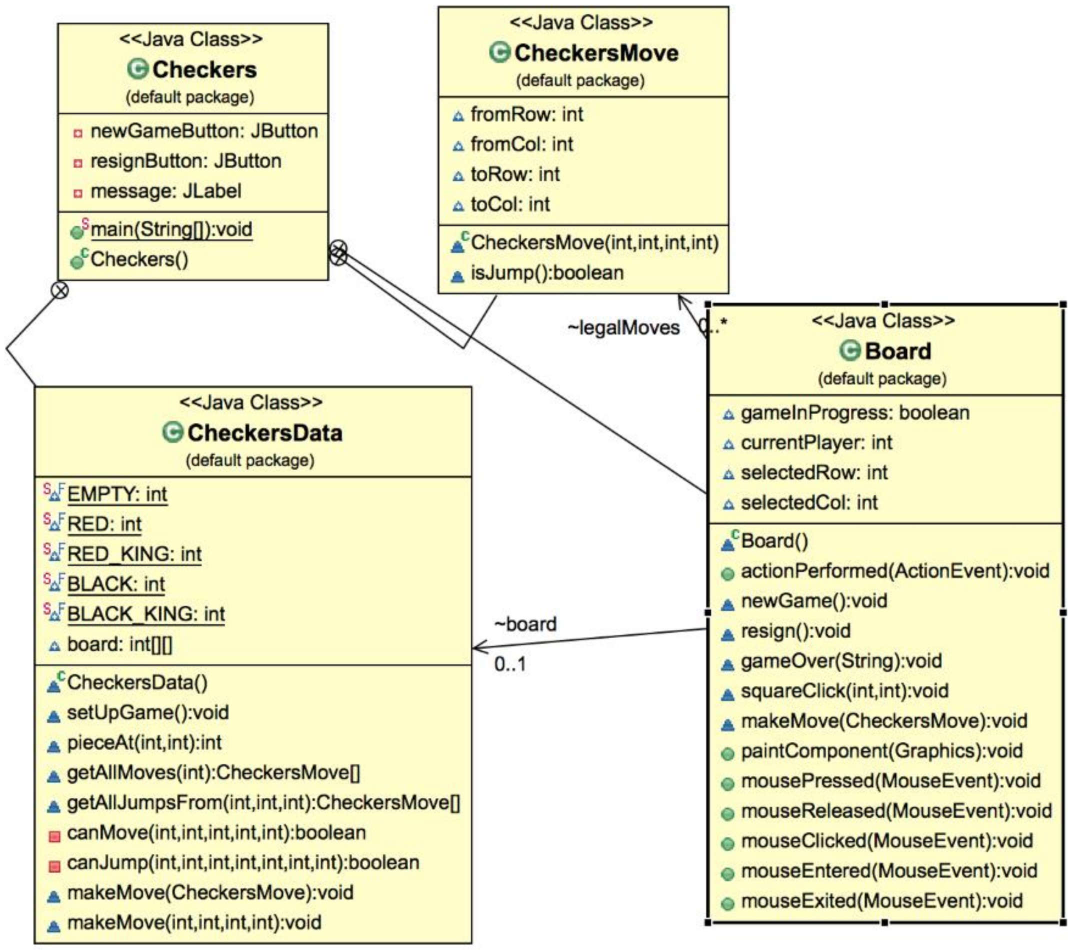
We used ObjectAid in eclipse to generate the following:



First Deliverable:

After writing the code for each game, we generated an UML using the eclipse Object Aid UML Explorer plug-in.

Checkers



Development

Final Deliverable:

For the final deliverable we met as a group when rewriting checkers. We followed observer patterns when writing the program. We wrote the board so it displayed the pieces first, we then focused on getting the pieces to move, after that we worked on the getting the pieces to jump correctly which took the majority of our time which made us come to the conclusion to create a debug mode to help fix these issues. Finally we added the game modes as well as the AI and finished up the remaining portions of the project.

First Deliverable:

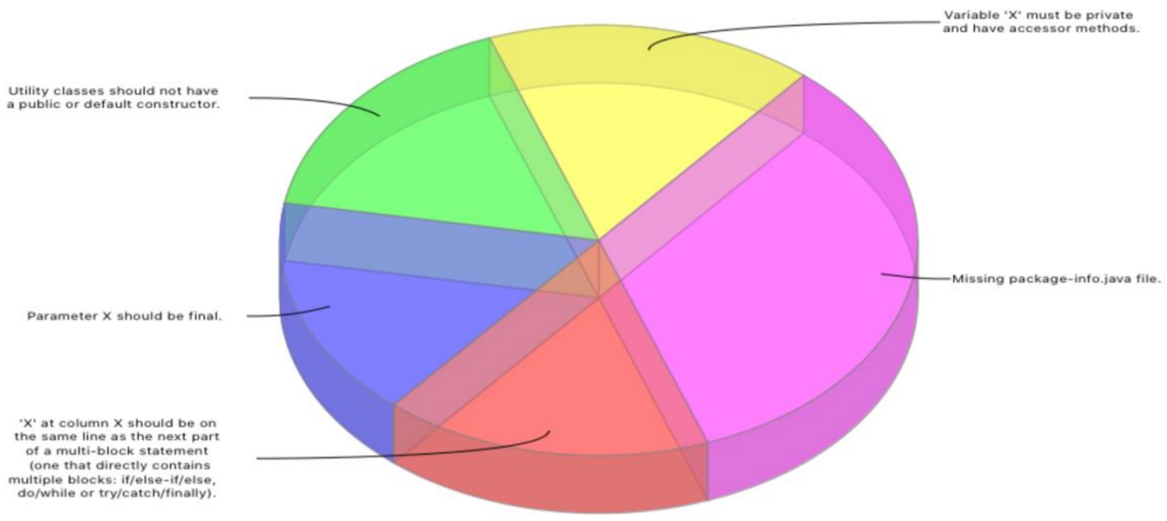
Team one was the one in charge of the development for the first release. For this part, we did not follow any specific style or standard, preferring development speed over quality code, which is the team two's job to prepare a suite of utilities for the second part of the project.
However, we still used a simplified Javadoc (with no specific flags) for documentation, ECLEmma for code coverage (not 100%), and checkstyle to check for the style, fully expecting to get a lot of style errors, since that was not one of our objectives with this first release.
For the future, the customer can expect a full object-oriented program, with code recycling through encapsulation, and polymorphism, by having a fixed number of classes inherited throughout the games, and easier Junit testing due testing code once.

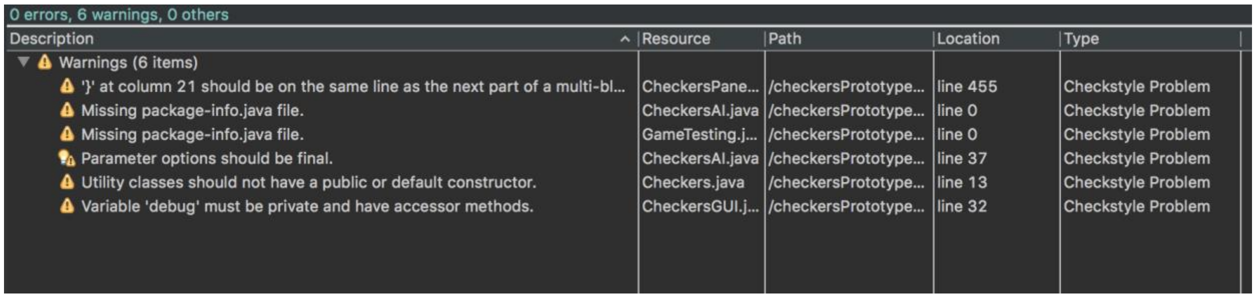
Code Standards

Final Deliverable:

There are a total of 6 errors from checkstyle:

- The first one is an error that we decided to not fix, because the else-if statement following the '}' has comments before it, and we needed the space; it was either having this error, or having more than 80 characters on a single line, which is another error recognized by checkstyle.
- The second and third errors are both because of a missing package-info.java file: we decided not to fix this two problems because we did not plan to create the package-info.java file.
- The next problem is a problem in the checkstyle preferences: in every computer we had, checkstyle would not use the preferences that are listed on blackboard, like ignoring magic numbers and the design for extensions options. We changed almost every parameter in the method definition to final, because it would have not changed our program. However, in this instance, option is changed inside the method itself, and could not be defined as final.
- The last two are non-errors: the first is a problem with the class containing the main method. In the class there are no constructors, but checkstyle is still asking to delete them. The second one is about checkstyle telling us to change a static variable to private and make a constructor. However, we followed a precise designing decision when making that variable static, and should not be changed.





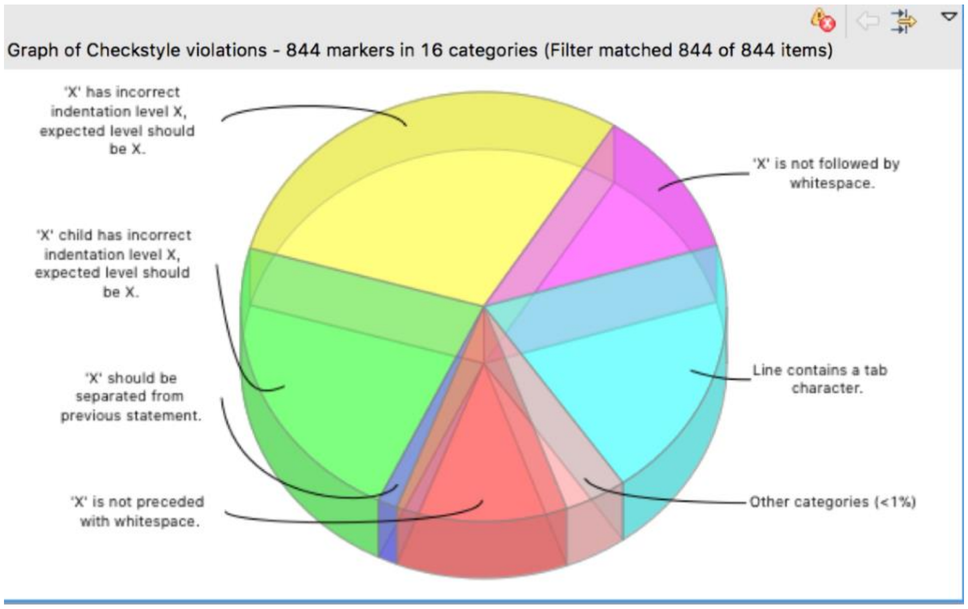
0 errors, 6 warnings, 0 others

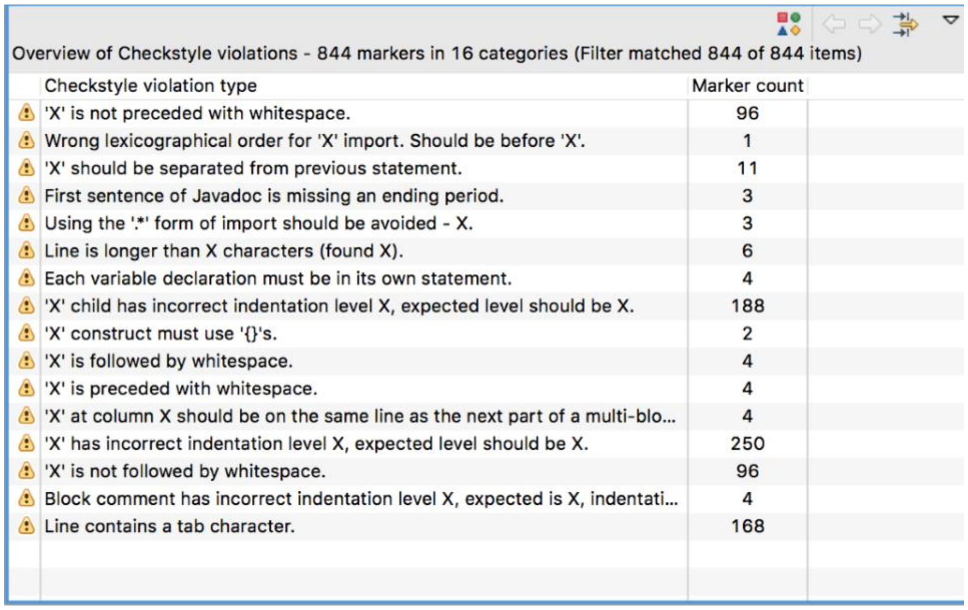
Description	Resource	Path	Location	Type
Warnings (6 items)				
'}' at column 21 should be on the same line as the next part of a multi-bl...	CheckersPane...	/checkersPrototype...	line 455	Checkstyle Problem
Missing package-info.java file.	CheckersAI.java	/checkersPrototype...	line 0	Checkstyle Problem
Missing package-info.java file.	GameTesting.j...	/checkersPrototype...	line 0	Checkstyle Problem
Parameter options should be final.	CheckersAI.java	/checkersPrototype...	line 37	Checkstyle Problem
Utility classes should not have a public or default constructor.	Checkers.java	/checkersPrototype...	line 13	Checkstyle Problem
Variable 'debug' must be private and have accessor methods.	CheckersGUI.j...	/checkersPrototype...	line 32	Checkstyle Problem

First Deliverable:

Checkers

This has many different check style errors that will be fixed when implementing this to a more object-oriented programming style. We mainly wanted to focus on the logic of the project for the first deliverance and making sure we had functional code, now we will make it quality code. The majority of these bugs do come from indentation errors which are caused by using tabs instead of spaces. This type of error can be ignored and the settings for check style will be updated.



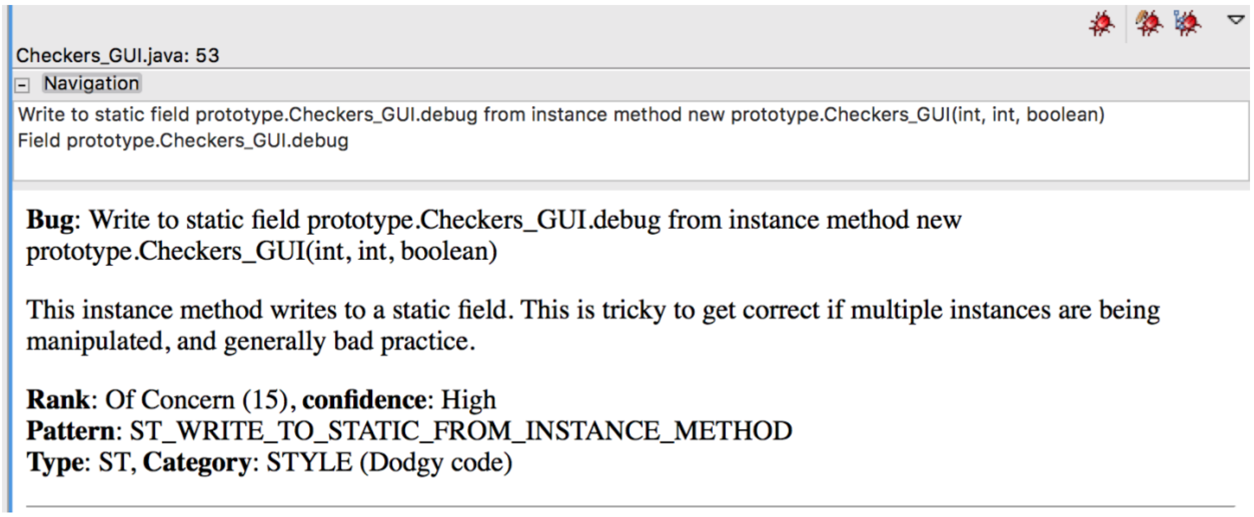

Overview of Checkstyle violations - 844 markers in 16 categories (Filter matched 844 of 844 items)

Checkstyle violation type	Marker count
'X' is not preceded with whitespace.	96
Wrong lexicographical order for 'X' import. Should be before 'X'.	1
'X' should be separated from previous statement.	11
First sentence of Javadoc is missing an ending period.	3
Using the '.*' form of import should be avoided - X.	3
Line is longer than X characters (found X).	6
Each variable declaration must be in its own statement.	4
'X' child has incorrect indentation level X, expected level should be X.	188
'X' construct must use '{}'s.	2
'X' is followed by whitespace.	4
'X' is preceded with whitespace.	4
'X' at column X should be on the same line as the next part of a multi-blo...	4
'X' has incorrect indentation level X, expected level should be X.	250
'X' is not followed by whitespace.	96
Block comment has incorrect indentation level X, expected is X, indentati...	4
Line contains a tab character.	168

Static Analysis

Final Deliverable:

When running FindBugs we determined we have one possible error. This bug does not affect the code it is referencing a static variable. It is showing as a bug because if we have multiple instances of the game playing it would change all of the instances and only affects the debug mode.


Checkers_GUI.java: 53

Navigation

Write to static field prototype.Checkers_GUI.debug from instance method new prototype.Checkers_GUI(int, int, boolean)
Field prototype.Checkers_GUI.debug

Bug: Write to static field prototype.Checkers_GUI.debug from instance method new prototype.Checkers_GUI(int, int, boolean)

This instance method writes to a static field. This is tricky to get correct if multiple instances are being manipulated, and generally bad practice.

Rank: Of Concern (15), **confidence**: High
Pattern: ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD
Type: ST, **Category**: STYLE (Dodgy code)

First Deliverable:

No Static Analysis with FindBugs available for the first deliverable, check the postmortem for the reason.

Code Documentation

Final Deliverable:

Everything is on the master branch. The prototype folder is the game's package, the testing folder are the JUnits, and the doc folder contains the generated JavaDocs:

https://github.com/GionataB/Project_CIS350.git

Initially, the code and the comments were on two different branches, so that different people could work on the code while the rest were working on the comments. However, we merged everything in the master branch for the final deliverable.

First Deliverable:

Minesweeper:

https://github.com/GionataB/Project_CIS350/tree/First_Release/minesweeper/doc

Everything is documented, but most of the comments do not contain flags. Including flags would have made the documentation better, but the first release is an old version and we are already moving to a new coding style, that would make our flags not up to date while the actual description for a method is still good.

Checkers does not contain Javadoc comments for classes, even the nested ones, because everything is part of one single file and we decided to not include redundant comments.

Configuration Management

For the final deliverable, we mostly worked together on either Gionata's or Brendan's computers. For that reason, the commit history on the master branch will show only those two. However, we also made branches for comments, and those will be put here as well.

Commit history
Master Branch:

https://github.com/GionataB/Project_CIS350/commits/master

Commented branches:

https://github.com/GionataB/Project_CIS350/commits/comments_final

https://github.com/GionataB/Project_CIS350/commits/checkstyle

https://github.com/GionataB/Project_CIS350/commits/Brendan

In the end, the master branch contains all the project files. Along with the Junit, the source code, and the documentation, the branch contains the files used to generate an executable .jar file. The testing and the .jar file have been done on a system with the java version 1.8.0_112 (obtained through the command java -version).

Deliverable:

https://github.com/GionataB/Project_CIS350.git

Verification

Final Deliverable:

For everything that is in the GUI we created the debug mode while everything else was testing using JUnit testing. Basically, all GUI elements, and all game elements that could not be tested without the GUI, have been tested through Junits. An interesting

note is that the Junit was more useful to let us know that the game was doing what we expected it to do, but manual testing showed use 5 to 6 possible exploits, that we fixed. One such exploit was: the white player selects a checker, but does not move. Then, concede the game. When the new game starts, the selected checker in the last game could now move everywhere in the board, just selecting a tile would make the checker teleport there. Another such exploit would be to “block” all enemy checkers: initially, the game would throw a lot of exceptions, but we fixed it.
Lastly, we recognize that the debug mode generates so many outputs that constantly playing with it would freeze the game, due the timer thread not being able to process the next event because the buffer would be full of system messages to dispatch. While there is no real solution, a fix is to use the debug mode only when testing something, and not for playing an entire game with it.

First Deliverable:

At the current stage of the project we have not done any unit testing because of the agile method we are using we will do our testing once the project is completely finished. This is because of time constraints as well as wanting to have the entire project functioning before we test.

Integration Tests

Final Deliverable:

We created a debug mode, in which we tested each individual feature by displaying the stats within the console and the GUI, to see that the GUI would correctly display the back-end code’s state.

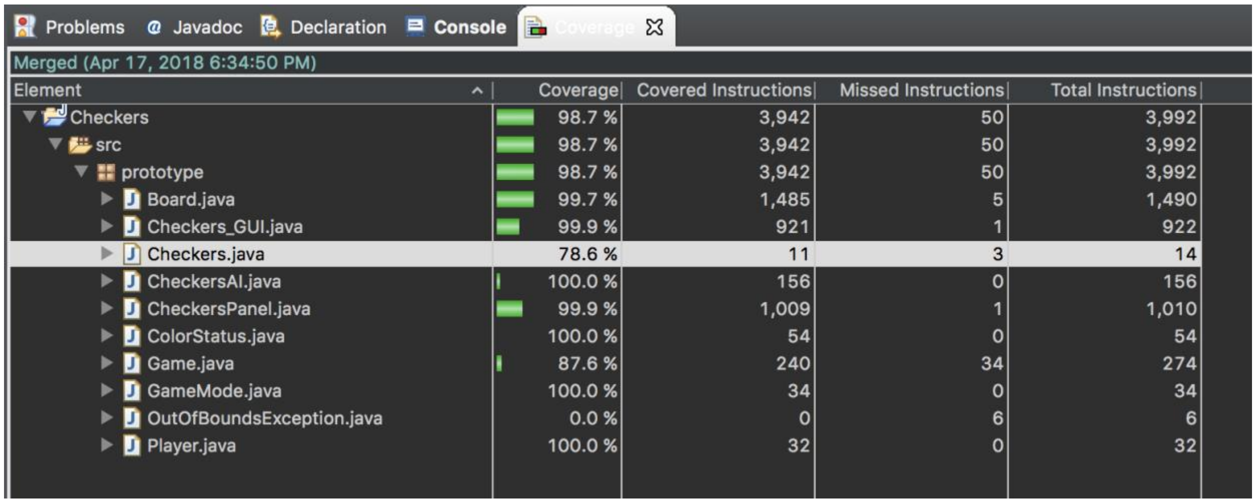
First Deliverable:

No integration testing in the code, refer to postmortem for further information.

Unit Tests

Junit testing has been done for the following classes: Player, Game, Board, CheckersAI.
Board and ChecekrsAI have been tested together, however due not being able to manually change the state of the board, we were able to test only half of the actual board methods.
The GUI helped testing the rest, since there is a direct representation of the board’s state on the screen. Game has been tested in its entirety, excepts the methods for the game’s timer (easily tested through the GUI) and inherited methods from java libraries, which we assume are mostly bug free and already tested. Player and AI have been completely tested.

Code Coverage:



Problems | Javadoc | Declaration | Console | Coverage

Merged (Apr 17, 2018 6:34:50 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Checkers	98.7 %	3,942	50	3,992
src	98.7 %	3,942	50	3,992
prototype	98.7 %	3,942	50	3,992
Board.java	99.7 %	1,485	5	1,490
Checkers_GUI.java	99.9 %	921	1	922
Checkers.java	78.6 %	11	3	14
CheckersAI.java	100.0 %	156	0	156
CheckersPanel.java	99.9 %	1,009	1	1,010
ColorStatus.java	100.0 %	54	0	54
Game.java	87.6 %	240	34	274
GameMode.java	100.0 %	34	0	34
OutOfBoundsException.java	0.0 %	0	6	6
Player.java	100.0 %	32	0	32

We are running at 98.7% code coverage, the reason we do not have 100% is because we have some constructors in the game.java that are never hit. Checkers.java the class header is never hit because an object is never instantiated, since the main method is called statically without an object instance.
The outOfBoundException.java is not used, but since most of the checks in the game are through the GUI, we feel it's important to have it if someone else wants to expand our code. Game.java has return statements that will never be hit and switch cases with default that will never be reached: that is expected because we want to be as precise as possible whether an if should be entered or not, to prevent careless errors.

Requirements Coverage:

Final Deliverable:

We met all of our requirements we decided to make for the second deliverable. We completed the game modes as well as the AI. We also added in the debug mode which was not the initial plan but determined was needed after we had errors within our jump method. This debug mode became our most useful tool for fixing any logic issues that occured while creating the different modes.

First Deliverable:

We have not done this part yet. Our requirements for the first deliverable have been met, but the final requirements are still changing.

Postmortem

Final Deliverable:

Self-Reflection

Gionata: We certainly worked better as a group this time. Funny enough, for the final deliverable we decided to focus more on programming and less on designing, because we found out we spent most of our time discussing over nothing and changing something that we didn't even implement yet. This way we worked better, made faster decisions, and created what we think is a good product. It may not be an extensive product with a lot of functions, but we put care in polishing it to the utmost of our capabilities.

Cole: After the first deliverable we all sat down and decided we did not have enough time to implement what we needed to with the time we had. We decided to just do checkers with multiple game modes and an AI. This project brought all of us to our breaking points and made us all hate each other at some point. The final product ended up being what we expected it to be. We all learned quite a bit when it came to work as a team. Overall it was a improvement form first deliverable and we worked a lot better together this time.

Rosa - After taking the time to sit down, draw out a design and create a set schedule that works for everyone, this second half of our project went significantly smoother. Also we may have decided to drop minesweeper and other games, however focusing on checkers let us really create a high quality product over quantity of programs. In addition to this, we were all able to partially learn some about artificial intelligence and creating a bot to play against a user, which in turn seemed more useful and educational then creating another game using similar logic to checkers. As a group we worked together very well this time around.

Brendan: This project went significantly better after the first deliverable. We got a way better schedule, we cooperated much more effectively, and were overall much more efficient. Working on solely checkers was a vast improvement and it allowed us to be much more effective in our quality of production. I am way happier with the second release.

First Deliverable:

Self-Reflection

Gionata: I dislike projects made in groups. It is not a matter of who I am working with, but how the project always ends up. A continuous change of ideas, troublesome communication and I always have the impression that the product is not what I'd like it to be. However, I think this is a great learning experience, it shows us the difference that unknown variables can make in what, at first, looks like a perfect design and schedule. As a side note, I have the feeling that when working in a group people tend to underestimate what they are doing, always thinking that there are others that can back up their shortcomings, which is partially ok, but it is a dangerous slippery slope that can endanger the budget and project as a whole.

Cole: Working in groups tends to be a hard concept to grasp when it comes to everyone's busy schedules, I personally enjoy working in groups that function well together. This group has had our struggles because it seems as if we all have opposite schedules and finding time to meet is sometimes impossible. The is project in general for the amount of time we have spent together is coming along, a little slower than expected but it is making progress. Hopefully after this first deliverable we will figure out more times to meet and make this program not only complete but also efficient.

Rosa: Overall as always working in larger groups has its usual challenges such as communication and scheduling but we have worked around these and overcome them. For the actual project itself I think we had no problem creating the code for our deliverables but had underestimated the amount of work documenting and testing would take. My last take away from this group is that although we had split up in the beginning who would work on what games and what code, it could have been very helpful for us to have also assigned documentation parts for each person. This could have allowed us to lay out what documentation would need to be done and by what dates, so that despite our clashing schedules people could be held more accountable for getting their parts done on time and/or at all.

Brendan: As with any group project, communication and management are prominent issues. This project is no exception. With many scheduling issues and a shortage in planning, – due to the scheduling issues – the work was much more difficult than it should have been. However, much of what we need to do now is uniting our disparate parts and focusing on creating a cohesive program that we can be proud of. With this goal in mind, hopefully we can meet more and develop a clear plan moving forward for the final release. From this point, communication will be even more crucial but I am confident we can take the many aspects of our program and make them work together seamlessly.

Group Reflection:

Final Deliverable:

We had our issues in the beginning, but they seemed to dissolve as we moved forward, from first to last deliverable we worked better together and more efficiently. Overall as a term project this team has completed work that I'm sure all of us are proud of. We decided to switch from a move individual approach to a group and it worked. We are all busy with other classes which proved extremely difficult, but we provided what we set out to do as efficiently as we possibly could with many more meetings.

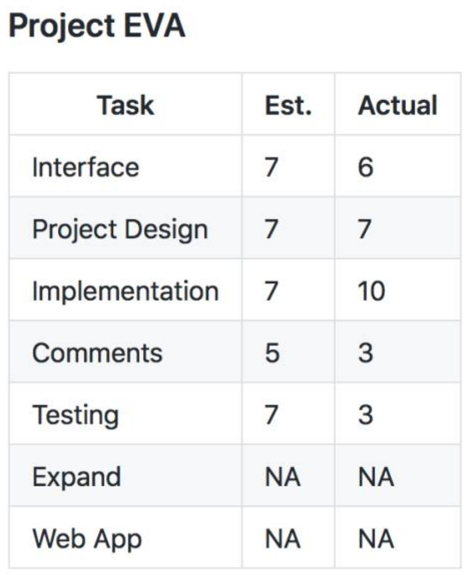
First Deliverable:

The major problem we found at the beginning was finding a day for a meeting and discuss the project. Since we couldn’t find a day available for every one of us, we decided to make two meetings, so that everyone was up to date on the project. However, due the difficulties encountered on working as a team, we decided to split the work in a way that every teammate is entirely responsible of one game. While this permitted us to advance extremely fast on the games, the end result was a mixture of games that had nothing in common, making the creation of a unified GUI and Junit testing especially hard.
For that reason, we decided to proceed with the code we have, and present a first deliverable that works, but that does not encompass all the project requirements.
Recently, we decided to split in two teams right after the first deliverable is due, teams made up of two people each: the objective of each team is to convert the two games presented in the first deliverable in code that uses the classes made by Gionata, so that Brendan can use his GUI on all the games, and the testing is much easier, and the JUnits much faster to write.

As a final note, our team originally had five members, with the fifth being Juan working on the Sudoku game. Unfortunately, after a few we lost all contact with him, and we never got any code. For that reason, we decided to proceed without him and that is the reason we did not include him in this paper.

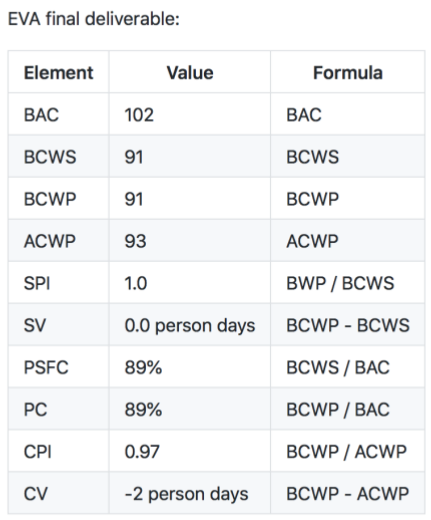
Earned Value

Final Deliverable:



Project EVA

Task	Est.	Actual
Interface	7	6
Project Design	7	7
Implementation	7	10
Comments	5	3
Testing	7	3
Expand	NA	NA
Web App	NA	NA

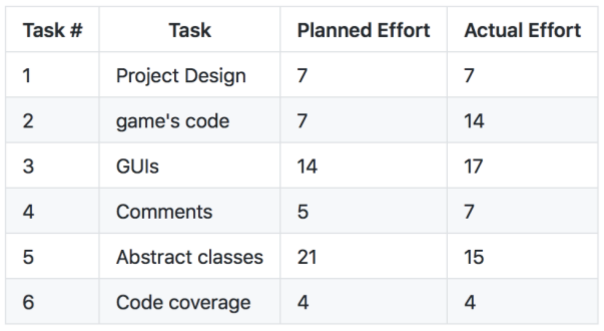


EVA final deliverable:

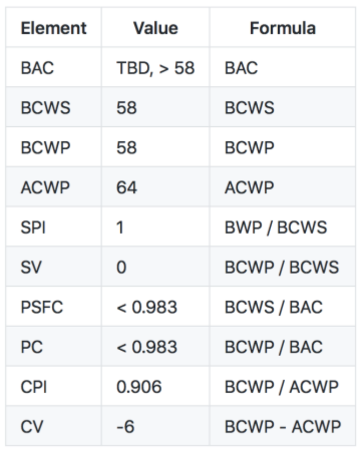
Element	Value	Formula
BAC	102	BAC
BCWS	91	BCWS
BCWP	91	BCWP
ACWP	93	ACWP
SPI	1.0	BWP / BCWS
SV	0.0 person days	BCWP - BCWS
PSFC	89%	BCWS / BAC
PC	89%	BCWP / BAC
CPI	0.97	BCWP / ACWP
CV	-2 person days	BCWP - ACWP

Since we decided to but Expand and Web App directly after the first release we cut the est. time that it also would have took. It From or final deliverable to this deliverable we did get out CV value back up to -2 instead of the -6 we were at during that deliverable.

First Deliverable:



Task #	Task	Planned Effort	Actual Effort
1	Project Design	7	7
2	game's code	7	14
3	GUIs	14	17
4	Comments	5	7
5	Abstract classes	21	15
6	Code coverage	4	4



Element	Value	Formula
BAC	TBD, > 58	BAC
BCWS	58	BCWS
BCWP	58	BCWP
ACWP	64	ACWP
SPI	1	BWP / BCWS
SV	0	BCWP / BCWS
PSFC	< 0.983	BCWS / BAC
PC	< 0.983	BCWP / BAC
CPI	0.906	BCWP / ACWP
CV	-6	BCWP - ACWP

In the Earned Values calculations, we made some assumptions: since not all the project’s tasks are defined, or have a defined budget, BCWS and BCWP are equal, because both based on the first deliverable, and BAC is at least larger than those two, so 59 at minimum. For that reason, any calculation involving BAC is an estimate, using the highest value that can’t realistically be reached. For example, PSFC is lower than 0.983. Numbers with decimal digits have 3 decimal places used.

Variances

Final Deliverable:

Warning: We have no warnings in our project due to adding flags to remove them, we had a few warnings that were serial UID warnings for Component objects. Since we were not interested in creating a binary file for the GUI as well, since we do not let the user change them, we preferred to suppress those warnings for all extended Component objects.

Coverage and testing: We created a debug mode in which we used to do our testing while writing the program then after the program was completed we wrote the JUnit tests for it.

Earned Value: Our earned value is slightly over budget, which is to be expected because we lost a team member and had to carry his slack when he left. We did catch up by 4 days though from our first to final deliverance.

First Deliverable:

Warnings
Checkers contains two warnings, both because they do not declare a final static variable from JPanel. That is expected and we are going to ignore those warnings.

Coverage and testing
We decided not to test the code for the first deliverable, since we are going to make substantial changes in the logic. Part of the reason is that testing right now would take too much effort and we do not have the budget to do it. For that reason, we decided to go a bit over budget to work on a system that will make testing much simpler and faster for the final version of the project. Code coverage has been made, but we do not have full code coverage due to some code covering a few worst-cases, or impossible-cases, due not having budget to test. This way, we are at least trying to present a bug free version of our games. FindBugs has not been used due to the useless information we would get from it: the reason is always the same, testing a code that will change is a budget expense we cannot afford at our current state.

Earned Value
Looking at the EVA, right now we are over budget for the first release. This is expected because part of our budget went towards preparing for the final release, so we expect to not go over budget in the final release. Or, in the worst-case scenario, we are not going to go more over budget than we already are.

Lessons Learned

Final Deliverable:

We learned quite a bit from this project, first of all we figured out just how hard it is to work as a group. We found it hard to meet with many of us having jobs as well as taking large amounts of credits. Everybody also has different skill sets where some people knew more when it came to development and we utilized those skills when it came to debug and developing this project.

First Deliverable:

We certainly overestimated the time it would take to make the actual games and underestimated heavily the time we should have used to design the project. While everything worked out in the end and we think we were right in the choices we made, it all worked out just because of the total time allocated for the project, that let us have enough room to go back and change our approach for the second deliverable.
All of us were used to spend the most time on writing the actual code, but for this project we learned that taking enough time to write down a good project design for everyone to follow would certainly save the most time in the end.