C++ Submission Manual

# Feature Report

## Extra Features

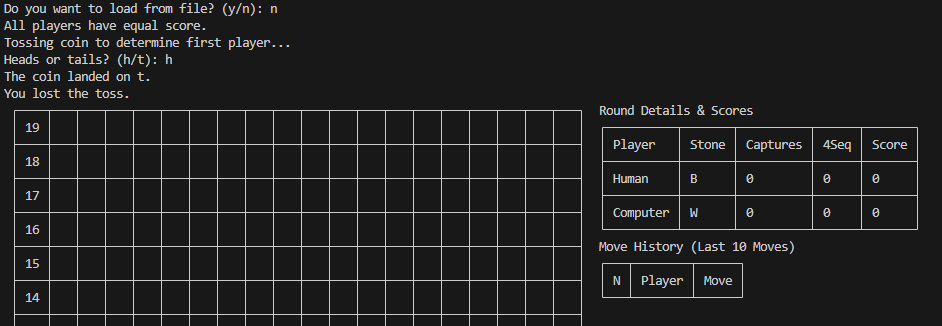
* Test-driven development: All functionalities that do not have any I/O are tested with full coverage – tests were written before the source code. However, once the interface implementation started, I stopped because I did not want to spend hours automating IO in the console and learning how to do it.
* Complete interface segregation: All inputs and outputs are separated from the logic of the game and the tournament. Only the Display classes (BoardDisplay, TournamentAnnouncement, RoundDisplay) execute output functions, and only the Human class takes input from the console. The main.cpp file includes other IO options such as reading and writing the serial file to the disk (the Serial class is responsible for generating and parsing the serial string and main uses that). Therefore, it is extremely easy to implement a GUI, a different console interface, or a different save file location/structure if it is needed.[[1]](#footnote-0)
* The board is displayed in a tabular box format along with round details and scores and moves history
* The board can be resized into any length and width, even rectangles. The length of the winning sequence can also be changed, so the board is compatible with any m,n,k game. For now, the capture length of two stones is a constant inside the board class, but that can be easily changed for a variable length. Everything works for different board sizes, except for reading serial files; writing serials still works.
* If there is more than one winning move, it finds the move that leads to the highest score and makes that move. However, if all moves lead to the same score, it holds back the winning move to accumulate more points.
* When building initiative, it looks two plies ahead. Using an exhaustive search was not feasible because it took too long. So, a depth-first search is used to find the neighboring cells of a sequence, and it only checks those cells. This speeds up execution and never builds initiative that will lead to capture.
* The game state is contained within the Board class (without any knowledge of who the players are and what their stones are). This enables the use of algorithms such as minimax with only one parameter. This also makes the game safe: the move is not made by the Player class – it is made by the board and the Round class gets the move from the player and instructs the board class to make the move. This prevents a rogue player class from making multiple/invalid moves
* No dependent class variables (except for is\_drawn and is\_won in the Round class); I am mentioning this because even the variable that specifies the current player is not used, because, in a valid game, the next player can be determined by the number of pieces on the board and the number of captured pairs. This, unfortunately, became a bug during the first demo because the provided serial files did not represent valid game states, and in haste to do the demo and make the program accept the serial files, I changed the number of captured pairs in the serial file, which messed up the demonstration and the evaluation.
* Pente library - The source consists of a library and an executable. The library can be used by any other executable that might have a different interface/ logic. With minimal refactoring the pente library can be refactored into a m,n,k library that can be used for any m,n,k game.
* Cross-platform - The code can run on any operating system. The build is defined by the CMakeLists.txt file with platform-agnostic configuration so that it is highly portable.
* All inputs, including serial files, are validated. Initially, the program would not allow the user to load invalid files, but now it displays a message saying the board is invalid, why it is not valid, and loads it anyway.

## Missing Feature

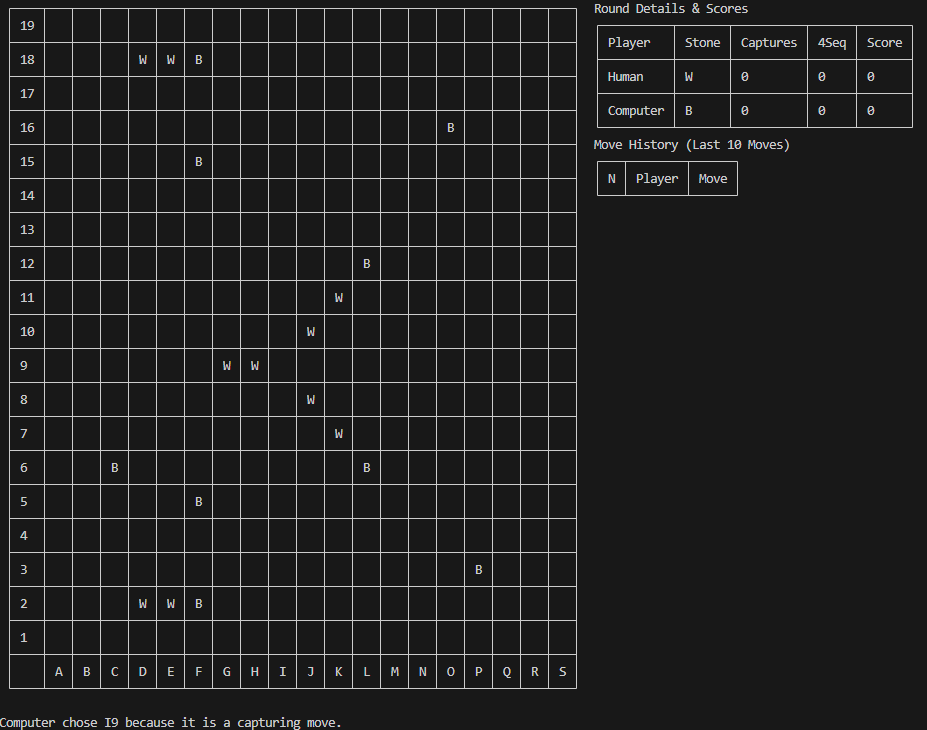
* Comments – Unfortunately, I did not have the time to comment on the technical design inside the source files. Wherever possible, I have refactored code that would need a comment to explain into a function with a descriptive name that explains what the code is doing.

# Screenshots

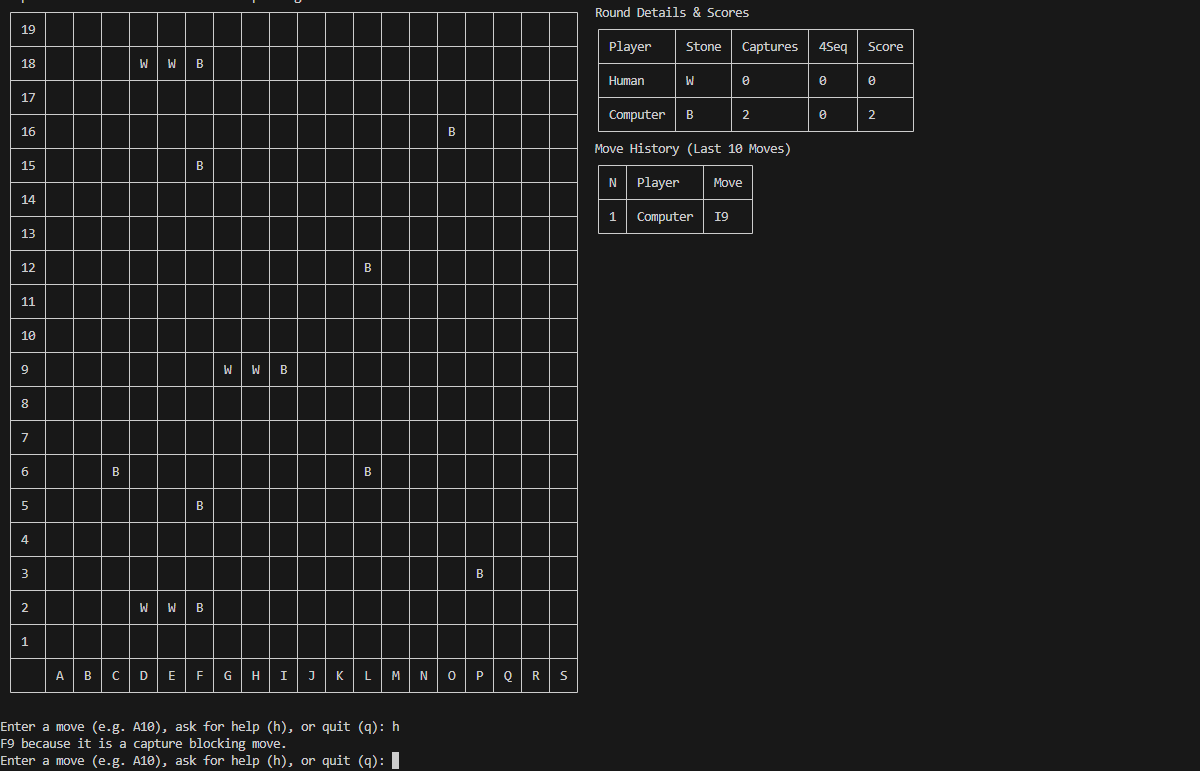
## First player of the round being determined



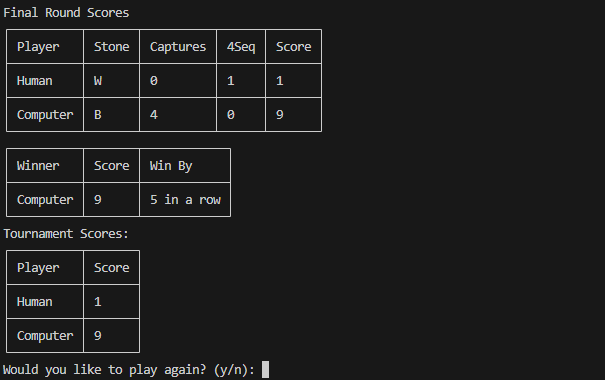
## Computer’s move being explained



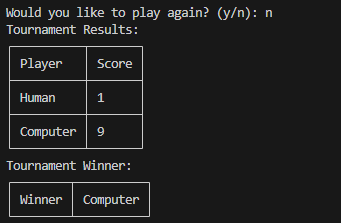
## Computer providing help



## Winner of the round being announced



## Winner of the tournament being announced



# Description of Classes

| **Class** | **Description** |
| --- | --- |
| Board | The board used for the game |
| Experimental  Board | Inherits from Board -- Allows the user to make invalid moves; Used for analyzing moves |
| BoardDisplay | Renders board in a tabular format with positions labeled |
| GameOver | Inherits from std::exception; Thrown to indicate that the game is over |
| GameDrawn | Inherits from GameOver |
| GameWon | Inherits from GameOver |
| InvalidMove | Inherits from std::exception; Thrown to indicate that an invalid move is being made on the board |
| Quit | Inherits from std:;exception; Thrown to indicate that player had decided to quit |
| MoveAnalysis | Provides analysis for a move, for example, if the move is a winning move, capturing move, capture blocking move, win blocking move, etc |
| Player | Represent a player -- virtual |
| Computer | Inherits Player |
| Human | Inherits Player |
| Position | Represents a position in the board; Class used because it contains methods to generate string representations and calculate distances |
| Roster | Stores the users and the scores in a tournament |
| Round | Represents a round in the game |
| RoundDisplay | Renders and display the board, details, and move history while a round is played |
| Serial | Parses and generates serial strings |
| Strategy | Decides on which move to play -- separated into a class from the player so that the same player can use multiple types of strategies |
| Tournament | Represents the Tournament |
| Tournament  Announcement | Renders and displays tournament results in a tabular format |

# Data Structures

| **Name/ Structure named defined using “using”** | **Description** |
| --- | --- |
| BoardSequence | Used to store full or partial parts of the board; evaluates to std::vector<char> |
| MoveHistory | Used to save the history of moves of a player; evaluates to std::vector<std::pair<Player\*, Position>> |
| analysis | A tuple to store the analysis of a move -- evaluates to  std::tuple<int, int, int, int, int, int, Score, Score, int, std::string>  Contains information about whether it is a winning move or not, whether the opponent can win, the differential in score after the move, the differential in score if opponent makes the move, the capture differential, opponent capture differential, and whether opponent can capture two plies ahead, the psedo score, the negative distance the center and the position  Initial plan was to create a class, but I ran out of time; A vector of this tuple is then sorted to find the most optimal move |
| priority queues | Usesd to store the move and their score for different types of moves; this was later discarded for one single tuple instead of having multiple queues and if blocks |
| StrategicMove | evaluates to a std::pair<Position, std::string>; the second string is the rationale for the move |
| map | Used to player to stones, stones to number of captures, and player to score |
| vector | Used for storing various other data structures -- no arrays used |
| set | Used to return list of moves -- for example, available moves, winning moves, capturing moves, etc |
| Table | Used to represent board, scores, and history for display -- from tabulate |
| graph | Not explicitly used in the code, but MoveAnalysis and Board use graph algorithms for various methods |

# Project Log

## September 1

* Setup IDE and development environment: Visual Studio Code is used with C++ extensions. The development kit is clang 10.0.0; cmake version is 3.25 and c++ standard is c++ 20 0.5 hrs
* Decided on the clang-tidy format. Researched which type of pointer alignment to use and settled with left alignment (towards the data) and which brace style (settled on the allman). Created a .clang-tidy format to get consisted formatting 1 hr
* Set up google test framework. This took me longer than anticipated (about 1.5 hrs)

Total: 3 hrs

## September 2-3

* Set up the project structure so that the tests and the library structure are clear and the build is done properly. Spent more than 7-8 hours (I went through the entire cmake tutorial) trying to figure out how to define an os independent library in cmake. Apparently, “target\_include\_directories(pente\_src INTERFACE ${CMAKE\_CURRENT\_SOURCE\_DIR})” was needed to enable test executable to find the library
* I initially set the library to be SHARED so that compile time would be shorter for the test executable. It worked on Linux, but when I switched to Windows, it didn’t. Spent another 2-3 hours trying to figure out what went wrong. Apparently, for Windows, the path of the dll needed to be specified, which would make it os dependent, so switched to static. This took me another 2-3 hours of perusing cmake tutorials and stackoverflow

Total: 12 hours

## September 4

* Started testing and working on the SerialFile class and defined functions to get scores and pairs

Total: 1hr

### September 5

* Wrote test cases to test board string; File is not used for defining the test serials and the string is defined in the code to make the project path independent. I did some research on how to use relative imports but because the file will be compiled, and the compilation location is mostly IDE dependent, I did not want to get into the hassle of configuring different IDEs, so I just put the string representation in the test file. ~1 hr
* Refactored the test strings into fixtures for readability 0.75 hrs
* I had written tests to determine that the serial input was working properly by checking the center move, but it was incorrect in the website. It should have been J10, but it is K10 on the board. Changed the serial file after emailing Professor 3 hrs
* The board class uses a dictionary to store captured pairs now, so that the next player can be independently determined without having to do anything with the Player class 1.25 hrs
* Refactor redundancy in test and src codes 1 10 mins

6 hrs

## September 10

* Created a Position class and added methods to get and parse the string representations and the numerical ones. 1 hr
* Tested and implemented the get\_row, get\_col, and diagonal getting methods 3hr
* Realized that there were two types of diagonal and researched their names. Finally, settled on main and anti-diagonal naming conventions 1 hr
* The src code took relatively less time, but the test codes are taking more time

5 hrs

## September 11

* Test and implement getting the total number of stones played on the board and getting the next player’s turn based on that; as a result, there is no need to store the player’s turn in a variable 2hr
* Test and implement the Roster class with suitable methods. The human, computer and player classes are also defined. Using a string to reference the user in the score map, but this could lead to problems if multiple users have the same room. It seems I need to add a comparator function to player class to store it in a map. I am trying to not use pointers, but it seems that is unavoidable 2hr

4 hrs

## September 12

* Test and implement making moves on the board. While making moves player is not necessary because the next stone can be determined on the basis of of the no captured pairs and the no stones on the board 2hr
* Test and implement getting available moves. This ensures that the first and second moves are compliant with the rules. The testing is taking longer than the implementation, but the tests are worth it 3hr

5 hrs

## September 13

* Add fmt/format library to cmake. It is already included as a standard library in C++ 20 for the microsoft compiler, but not yet in the linux compiler. Again having to configure cmake to use the latest git repository is a headache. I would rather be coding.. C++ is annoying when preparing the build files…

3 hrs

## September 14

* Start working on BoardDisplay. Tried printing out the board like in the serial file, but it’s very difficult to see which stone is where. Tried formatting with + , - and | for the grid borders (which took a lot of time), but the width still looks inconsistent and hard to read with a lot of display space occupied on the board. 2 hr
* Started implementing a table display, but erased it because there must be a library for it.30 min
* Started searching for libraries to implement the table. This took me a lot longer than expected (~2 hrs). Finally found the tabulate library.
* Using the tabulate library to render boards, because it makes it much easier to see the board and identify the pieces 1.5 hr

6 hrs

### September 15

* Test and implement handling of captures up, down, left and right. The test code is taking a lot longer, but it really helps with the refactoring so can not give that up 2.5 h
* I am thinking of using a Range class if I have time in the future to get sequences from the board. Then mathematical transformations can be applied to the range to get different sequences from the board without using nested loops. But this will take longer than anticipated so skipping it.
* Test and implement handling of captures in the diagonal. Again the test takes a lot longer than the actual code ~ 10 times more, but that’s okay 3.5 hr
* The handling of capture is test using if conditions for all 8 different directions. Using a Range class to get the different sequences would be much more elegant, but I don’t have time for that
* Test multiple captures, so that all captures are taken into account. Deciding the test cases is taking a lot of time, but better be safe than sorry 1 hr
* The test cases were displaying the board so that I can see if the board was rendered correctly or for debugging purposes. I removed the displays and the tests ran ~ 100 ms faster. I thought it was the capturing algorithm that was taking so much time, but apparently std::couts consumes a lot of time… Also refactored redundant code 1 hr

8 hrs

## September 18

* Test and implement analysis of first and winning moves. A MoveAnalysis class is used to analyze the moves – it contains methods to determine how a move can change the board. Is initialized with a board and a move 2 hr
* Test and implement checking if the move is not a losing move. I initially called it win\_blocking\_move, but that would be a misnomer. It checks if the move would not lead to a loss 2hr
* Test and implement checking if the move is a capturing move 2hr

6 hrs

## September 19

* Start implementing the executable. So far, everything was done for the pente library and main did not exist until this point. All the tests were done by the testing framework. 15 min
* Main is decomposed and classes for Tournament, skeleton for Round and TournamentAnnoucement are defined. At this point, I am not writing the tests before the source code, so I am uncomfortable, but it would take a long long time to write tests for IO so skipping it. 5 hrs
* Refactored the Roster class to use player pointers instead of player names for the functions. I initially did not want to use pointers here but it reference variables do not work for inherited class and using the name of the player could lead to problems when multiple players have the same name (even though this would not be the case for this assignment) 30 min

7 hrs

## September 20

* Implement Round. I have maintained interface segregation throughout, so I did not want to use any console inputs and outputs inside round too. So round calls methods in other interfacing classes such as Human and Board, and they throw exceptions that Round catches. The Exceptions header file contains all exceptions, and I thought it would be much more elegant if the Exception file was not broken down into implementation because there are not many. Also the communication between game states such as game over, win, and draw are made between board and round through these exceptions. 5 hr
* Enable getting and handling moves from human. Moves are validated by creating a copy of board class and making the board. It’s easier to ask forgiveness than permission 2hr
* Make computer play random moves. This uses helper functions defined in their own files to make random moves. 0.5 hrs
* I initially did not use the std::random\_device, generator and uniform distribution, but that always led to tails for the toss. Had to research why it was not working and had to re-implement it. 3
* Apparently, template function implementations have to be in the header file otherwise it won’t link. I thought this was only for classes, but it is also for regular functions. This is problematic because class methods when using templates have a fixed number of possible data types they use, so they can be declared below the definition in the .cpp files, but picking a random element from a container in a helper function has to work for several use cases which are not feasible to list. Took a while to figure it out. Instead of importing header file into implementation, the implementation has to imported into the header. Not sure if this is worth separating into implementation and header files 3 hrs

12 hrs

## September 21

* Test and implement getting stone sequences from a board. This returns a vector of a vector chars that shows all the sequence in the board. This can be used for various functions like counting the number of 4s in a row for calculating score, for checking win and for determining optimal moves 3.5hrs
* Refactor all literals to symbolic constants. Almost everything was in symbolic constants from the beginning. However I was using 5 as the length of the winning sequence, so refactored that into a variable and made the board and logic compatible for any m,n,k game. Also used the “using” statements to make composite data types much easier to read. 0.5 hrs
* Did some major refactoring to reduce redundant code 0.5 hrs

5 hrs

## September 22

* Test and implement getting score for 4 consecutive stones and for 5 consecutive stones. This uses the get sequence function defined yesterday, so it is very elegant. 2 hrs
* Test and implement awarding points for capture 1 hr
* Realized there was a bug in the algorithm to get all main and get all anti diagonals. My rationale was we could get all main diagonals by getting the diagonals of all cells in the anti-diagonal, and vice versa but it clearly was not the case. It took me a long time to figure out the logical error in this. Now it goes throught the first row, and then through the first column to get all diagonals. 3 hrs

6 hrs

## September 23

* Test algorithm to get pseudo score that’s used for making moves. The pseudo score calculates the score for every move to determine its feasibility for various algorithms. Because the actual score is only helpful when there are 4 in a row or 5 in a row or for capture, it is not helpful when determining which move to make in the beginning. Currently, the pseudo score is the sum of the product of the number of sequences of length greater than 1 and the number of sequences. Ie. count \* length for every stone sequence in the board. 1.5 hrs
* Start implementing RoundDisplays. I don’t like the board display with the edges also having the ╀ . Wrote a display helper function to convert the table to use proper formatting for the edges. Took me more than 6 hours to get everything displaying properly and to figure out the tabulate library. Windows was also not rendering unicode and trying tow work around that but gave up after an hour of futile attempt to make the code cross platform and still renderable in windows. I should have been working on the computer algorithm instead. 7 hrs
* Implement history table, score tables, and tournament tables display 3 hrs
* Check if strategy works with a minimax algorithm. It takes too long for a 19 x 19 board – I waited for 1 minute before I stopped, and I don’t think alpha-beta pruning would have helped much either 4 hr

16 hours

## September 24

* The Strategy algorithm now checks if the move is a winning move, if its a opponent winning move, or if its a capturing move and the pseudo score. All moves that satisfy these criteria are stored in their priority queue and ordered on the basis of the score and the pseudo they will get. The highest-scoring move is popped off the queue. This then ensures that if there are multiple moves of the same type such as capturing or winning, it always chooses the most optimal one. If the two highest winning moves produce the same score, it holds off the winning move to get more sequences, but if there is a winning move that leads to more points than the others, it plays that move. 8 hrs
* The pseudo score algorithm is improved to take into consideration the score the opponent will get if they play a move there. Had to create an ExperimentalBoard class because the Board class would not allow any invalid move. Try working around it by making every possible move except that one and checking opponent's score, but this was too slow. This took me a lot of time. It’s much faster with the ExpermintalBoard class. Had to factor out some private methods to protected to enable this 3 hrs
* At this point all the requirements in the website are satisfied, but the algorithm is not still optimal enough to win every time

11 hrs

## September 25

* I wanted the algorithm to always win or draw, so I spent a lot of time playing it and trying to fix it. Used multiple factors like checking the average length from own stones and opponent stones, and distance from the center, but even then it did not always win. 5 hrs
* The algorithm only checks one move ahead, so when building sequences, it often makes a move that will lead to a capture. Checking two plies for every possible move took too long to run, so used a depth-first search to find all the neighboring empty cells and checked two plies ahead. Decreased execution time by a lot. 4 hrs
* Instead of using multiple priority queues, use a tuple to create a table of analyses that stores differentials for various types of moves (see analyses data type). The vector of tuple is then sorted to determine the most optimal move. The code length decrease by more than 50%, but the rationale for the move is not as detailed as it was previously. This reduces a lot of nested if statements and for loops, so the trade-off is worth it. 6hrs

15 hrs

1. When enabling, loading of invalid serial files, I put a console output inside the Board class. The optimal way would have been to raise an exception, and then call the loading function with a force (boolean) parameter if the error was detected. I unfortunately did not have time to refactor this. [↑](#footnote-ref-0)