# AP report

## Overview

For this project I was tasked with creating a multiplayer online curling game in C++. To achieve this I used the pool game we were provided in the workshops as a base as it already had all the openGL and collision functions setup. I have adapted this to create the curling sheets and be able to play a turn and team based game on each of the sheets. To produce multiple sheets I created a class which I then used to duplicate the sheet across the view field.

To run the program please open the threaded server solution file in visual studio found in the threaded server folder. Please run this file once it has opened, once it has opened to run on the same machine, please open the poolGame solution file in visual studio. The run this and input a name for your player, when it comes up with “HostName/IP:” please input ‘localhost’. And when it comes up with “Port:” please input the first port that was shown when you ran the server for the first user as it will be a master user. After that you can user the sockets in any order as they are treated the same. To move the cue use the arrow keys and press enter to shoot the stone. If you wish to change the number of clients please change the MAX\_NUM\_CLIENTS variable in the threaded server file.

A picture containing diagram

Description automatically generated

Figure - Game running with 5 sheets

## Implementations

### Stone Implementation

For the implementation of the stone class, it has to control the velocity and team of the stone. To implement this, it has a position and velocity to control the movement of the stone. These are implemented using the update function where the velocity decays based on the friction force and the position is updated with this new velocity. To start the stone moving it will use the ApplyImpulse() function which will set the velocity to the whatever the impulse is. This will then cause the update function to start moving the stone at the speed of the impulse with the decay of the speed with the friction force. This friction force is calculated by the ApplyFrictionForce() function which takes the current velocity and decays it using the current velocity as the negative acceleration. This causes the deceleration of the stone to decrease as the stone slows which adds the feeling of it sliding over the ice.

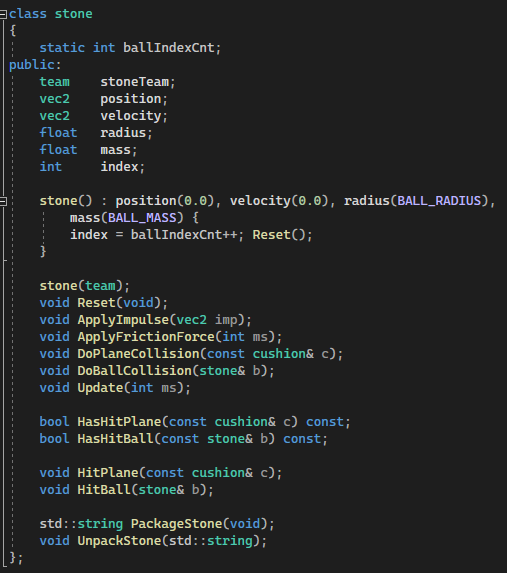


Figure 2 - Stone Class

### Sheet Implementation

For the implementation of the sheets I created a class which would then be used to create a single sheet with a variable called sheetNum which would hold the number of the sheet allowing it to be easily placed in a different location. To move the sheet I used the sheetNum with an equation of where x is the sheetNum. This equation means that when x is 0 the sheet gets placed in the middle position with the sheets then alternating left and right whilst moving out. This means that the sheet 0 will always be in the middle of the layout. The sheet consists of the 4 main walls around the outside which bounce the stones off them, the hog and hack lines which show where the stones are removed if they are not in between and the circles which make up the house. The class also stores the current stone count for easier loping through all the stones, a vector of the stones to allow pushback onto the end of the vector, the cushion and feature arrays which hold the graphics and information about all the drawn objects other than the stones, a particle set which holds the information about the particles, the hog and hack line positions and the centre of the house position. The cushion array holds 4 cushion class objects which contain all the information about the cushions for the game to draw and interact with them. The feature array holds a mixture of line and ring class objects both of which are children of the feature class. These arrays are used to draw the features of the sheet such as the hog and hack lines and the rings on the house. The hog and hack line positions are integer values which are used to test against the vector positions of the stones to see if an of them need to be removed. The centre of the house is a 2d vector which stores the position of the centre of the house which is used to score the stones at the end of a round.

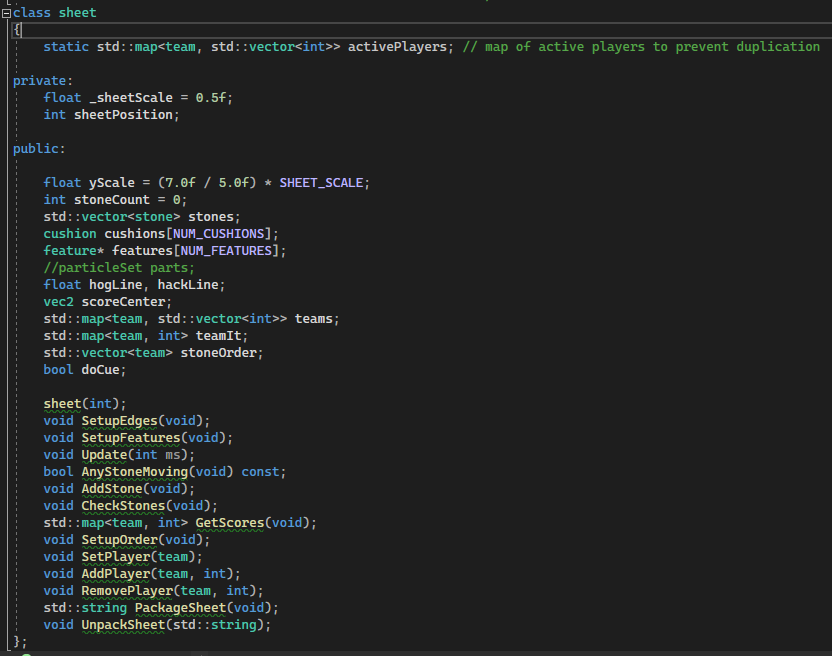


Figure 3 - Sheet Class

To create the cushions array the class uses the function SetupEdges() which manually sets all of the cushion dimensions and adds them to the cushions array. It also sets-up the centre and normal of all the cushions for the bounce of the stones. The SetupFeatures() sets-up all the extra visual features of the sheet including the hog and hack lines as well as storing their positions in the relevant variables and the rings for the house including storing the centre of the house in it’s variable. The AnyBallsMoving() function is a function to check if all the balls have stopped moving by looping through the balls vector and checking the velocities of each. This allows the main loop to easily check when the next stone should be spawned, when to check if the stones are past the hog and hack lines and when to allow the player control of the stone again. The CheckStones() function loops through the stones and removes any that are beyond the hog an hack lines. By looping trough from the last element this avoids the issue of it accessing an element that has been moved due to the removal of another stone or it skipping a stone that is beyond the lines and should be removed. The AddStone() function simply adds a stone to the starting position of the sheet and appends it to the vector of the stones, also incrementing the stone count by 1 to reflect the new count of stones. The GetScores() function is used to produce the scores of the sheet at the end of a round it uses the scoreCentre variable to calculate the distances of each of the stones and see if they are on the house. The update() function is used to check the position of the particles to see if any need to be removed from the scene to reduce the load on the computer of rendering many particles even if the camera can see them.

### Players, Team, and Scores

To implement these I have used classes for the teams and players and just stored the current score of each team in the team’s class. Figure 4 shows the header file for the player class I implemented, you can see the variables and function declared for the class. The ID is a variable to ensure that each player is different from each other, the IDcnt is a static variable that is incremented every time a player is created so that the ID is always different. The doCue is used to tell the player when they can aim and shoot the stone and the name is the name that the player submitted. The TableID tells the player which table it is at and is used for controlling the stone impulses. The Constructor declared creates a player with a randomly generated name, doCue being false and a tableID of -1 which means unassigned. The Package and Unpack functions are used to convert the class into a byte-like format to be sent over the network.

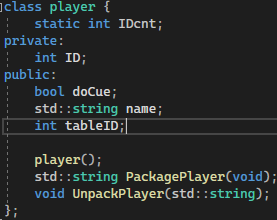


Figure 4 - Player Class

The team class implementation can be seen in figure 5, It has a static vector of activePlayers which was intended to prevent duplication of players but was causing issues so is currently of no use. The name is just the name of the team as with the player, the colour variable is a vec3 and is used to colour the stones on the sheets to separate the teams. The players vector is the vector of all the players on that team, and the currentScore is the score for that team. The pack and unpack functions do the same as with the player class converting it to be sent over the network. The AddPlayer functions adds a player to the team and the activePlayers vector to store the currently active players, the removePlayer function removes a player from the team they are on and the DeletePlayer function completely deletes the player from memory.

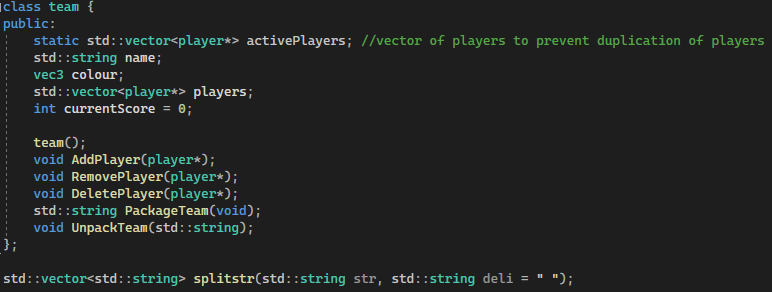


Figure 5 - Team Class

To implement the scoring I researched how the scoring in curling was calculated and found images such as figure 6. I also, looked at some articles about it can learnt that the team with the closest stone scores the amount of their teams on the house. The house is the rings which can be see in the figure, this is then cumulated over multiple rounds for the end score.

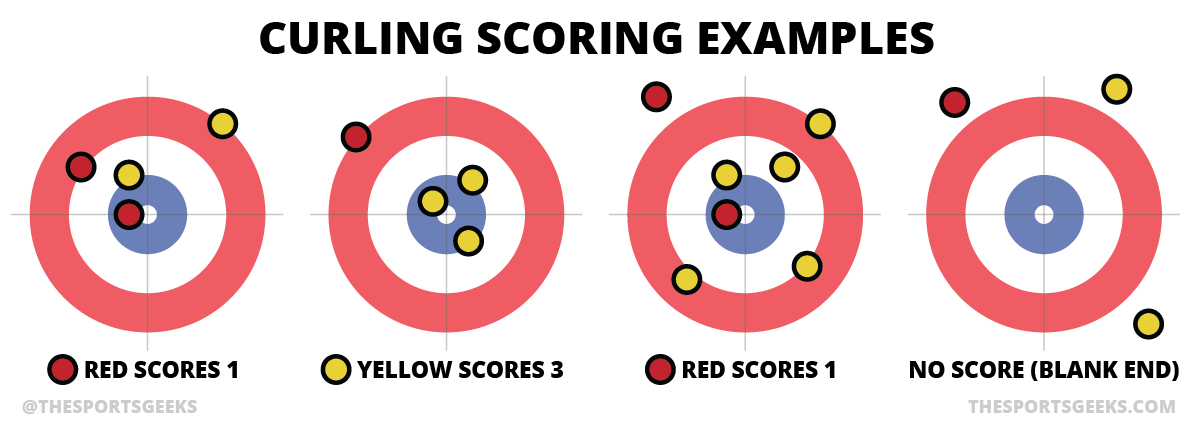


Figure 6 - curling scoring

In figure 7 you can see the code that I have implemented to calculate these scores. The first 4 lines are just declaring the variables I will be using, the scoreDict stores all the stones that are on the house by the team and the closest stores the current closest distance to the centre. The closestTeam is a variable for the current closest team to save looping through again. The first loop, loops through all the stones on the sheet and calculates there distance to the centre of the house and stores it in the scoreDict if it is close enough to score. If it is closer than the previous closest it store the team that the it was for and the distance it was. The cout statement outputs the winning team and how many points they scored, the second loop is to find the team in the gameManager class to assign the points to their team.

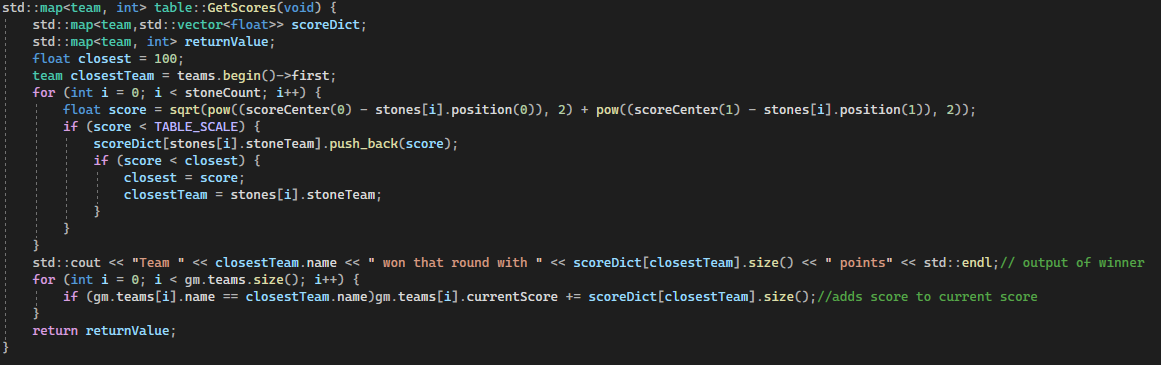


Figure 7 - Scoring Code

### Networking

To implement the networking I used the threaded server and client provided for us and adapted it to the task required. In figure 8 you can see the client class, I have added multiple functions to aide in the running of the program and have changed the getmsg() function greatly where when it receives certain instructions it will trigger parts of the code to write the current configuration and other commands. This allows the client to be listening without constantly requesting information instead just updating when it gets the new information. All of the send functions send information to the other users such as the impulse to apply or a new player to add to the team or a command to add a new stone.

In Figure 9 you can see the implementation of the interact function which is listening of the config command which is sent when a new player joins the server. When this command is heard it sends it to the 1st socket in the list and asks for a config which is then broadcast to all the users. All other messages are went to all users expect the sender and any commands done from them are done in the getmsg() function in the client.

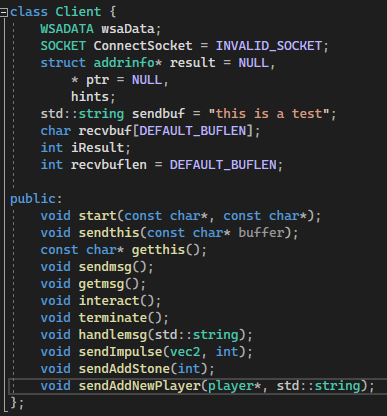


Figure - Client Class

Figure 9 - Server Interact function

In an idea world I would have put a gameManager object onto the server so that none of the clients have the ‘master’ copy instead the server does. This would make it easier to update the users on the current state of the sheets and would likely make it easier to scale up as the server could be run on a more powerful machine. Instead, the current solution will be throttled by the processing power of the ‘master’ client on the first socket.

# Comparison

I have developed this game using C++ however I could have used another programming language such as C# or python. Python is a much higher-level language than C++ being more abstract and often easier to read. This makes it much better for entry level and beginner programmers but can sacrifice useful features and options (Ateeq, Habib, Umer, & Muzammil, 2014). The loss of features means that it can be slower than C++ in many instances (Prechelt, 2000), especially in parallel instances, as the python GIL (global interpreter lock). The GIL prevents multiple threads from executing python bytecode in parallel which is required as Cpython’s memory management isn’t thread safe. Cpython can be multithreaded but only 1 thread can be executed at once which reduces the usefulness of multithreading. Some processes happen outside of the GIL and can utilise multithreading such as disk or network I/O and so multithreading is only a bottle neck for programs that have much of their execution time inside of the GIL. In comparison C++ allows full multithreading of the program relying on the developer to manage the threads and any race errors. It gives you many different methods to prevent race conditions and problems with shared data such as atomic functions and barriers. This allows the developer to take full advantage of multiple cores and speed up the processing of the program if it can be parallelised.

Another key difference is that C++ is a compiled language whereas python is an interpreted language. This means that C++ is converted into an assembly script which can then be run on the machine whereas python produces an output from the program. As python is an interpreted language the source code is the script file, and it is then interpreted by the interpreter on the machine. This often makes the program easier to debug as the interpreter often has extra debugging features that a compiled program can’t use. However, in general interpreted languages are slower than the compiled languages due to the extra interpreting done by the machine (Kwame, Martey, & Chris, 2017).

Python is often used by beginners as it has many useful features such as the automatic garbage collection. This means that any variable which has no references will be removed from memory by the garbage collector. This prevents a memory leak issue where a loop creates new variables which aren’t deleted and so take up memory even after they aren’t being used. C++ does have some automatic memory management with normal variables, where they are automatically removed when they go out of scope. However, with pointers this can cause a memory leak issue where the pointer is removed but the object it is referencing isn’t making it impossible to remove after the pointer has been removed. If this is done in a loop the program can quickly run out of memory causing it to crash due to the long response times due to the large amount of paging. If the programs memory leak is large enough or the programs memory isn’t limited it can cause performance issues on the entire computer (Xie & Aiken, 2005).

The last point of comparison is the networking, both of these languages use a socket-based library to connect and communicate between computers. Python uses its own socket library for the connection and transfer of data. For the data serialisation python can use JSON, XML or YAML which allows it to send data between programming languages as they are set forms for serialised data. Python also has the use of pickle which allows the direct serialisation of classes and complicated variables which can be useful when sending data between clients. C++ uses its own socket file with many of the same features as python, however it doesn’t have the use of pickle only JSON, XML or YAML. This makes it slightly more complicated to send complex data between programs as it will need a transition function to convert the data to a serializable format.

In summary C++ is a much more controllable programming language opening up the option to manipulate almost everything the program is doing but it is much harder to learn to use at its full capacity. Also, poorly written code in C++ can cause performance degradation with the issue of memory leakage and other issues such as race errors while multithreading. Python however, is easier to developing being more readable in many circumstances and being an interpreted langue makes the debugging much simpler in many cases. But, this means that there is a loss of performance compared to C++ and can mean it is less scalable as the users gets very large. However, as (Prechelt, 2000) say the differences in the performance are often smaller than the differences between the ability of the programmer.

## Scalability and Memory Issues

There will likely be many issues with the program if it is scaled up to a large degree as there aren’t many features implemented to handle large processing loads. It is unlikely that the networking will be an issue, I would assume that the first issue would come from drawing large amounts of objects every time step. The network will likely not be an issue as the bottle neck would be the server and that can be easily placed on a more powerful machine. However, I will note that the server is running on a socket per client basis which is a waste or resources and will cause issues if it is expanded or push to a more professional position. This issue with the server can be solved by creating a more parallelised server as described by (Abdelkhalek & Bilas, 2004). In their testing they found that the bottle necks where the fine-grain interactions and long wait times due to synchronisation issues. These issues are unlikely to affect this project due to it simplicity and minimal interactions or time constrained actions form the clients. The main issues I can see arising are the memory issues with drawing and storing all the objects as there is no system to reduce the amount that is being drawn. If some of this storage is moved to the server this will decrease the workload for the client’s computer needing only one more powerful computer to run the server. It will also reduce the amount of maths done by the clients with it being done on the server instead and the clients only drawing for the user.

## Challenges and Solutions for large amounts of players

As discussed in the previous section, one of the main issues will be the storage and calculations required for large amounts of stones and sheets. To solve this issue, I would move most of the processing to the server such as the mechanics of the stones, the scoring and management of the game. This would one reduce the amount of duplicated maths that currently occurs and would secondly improve the consistency of the game as there is only one master copy on the server and the other clients just drawn from it. I would still keep the maths for the cue on the clients side to reduce the stuttering if the client has a poor connection and make it easier to draw quickly.

As discussed by (Diaconu, Keller, & Valero, 2013) in relation to Minecraft, which is a much more complex game, maintaining large numbers of players can be resource intensive even on the client’s computer. This is especially noticeable with large numbers of players as the client nor the server will drop any person or information even when not required at the minute. The solution found by (Diaconu, Keller, & Valero, 2013) was to use Kiwano to take the clients requests and switch them to different servers. So, instead of having a single server running and storing all the users it is spread across multiple. This allows them to have massive amounts of users on a seemingly single server with all the interactions but with the extra performance for the clients.

To reduce the processing time required for drawing all the sheets and stones I have begun to separate the drawing into functions to allow multithreading. This, will drastically improve the performance on more powerful machines as they will spend less time drawing the objects and can process more per second. This will help with the scaling as each sheet will take up less time to process reducing the impact of the larger amounts of sheets.

To further reduce the impact of the large amount of drawing I would implement a culling of non seen objects. So that no objects that can’t been seen by the camera will be drawn, you can see an example of this in Minecraft in figure 10. This shows the wireframes of all the objects that the player can’t see from the cameras perspective. This approach is widely adopted and can lead to significant performance improvements especially on less powerful machines. To implement this I would need to implement a fast algorithm to check if a object can be see by the player, but as the camera is stationary I can do the maths once to calculate the line outside of the view. And then just use fast maths to work out if an object is inside or outside of the view of the player.



Figure - Minecraft Culling

# Bibliography

Abdelkhalek, A., & Bilas, A. (2004). Parallelization and performance of interactive multiplayer game servers. *International Parallel and Distributed Processing Symposium* (pp. 72-). Santa Fe: IEEE.

Ateeq, M., Habib, H., Umer, A., & Muzammil, R. U. (2014). C++ or Python? Which One to Begin with: A Learner's Perspective. *2014 International Conference on Teaching and Learning in Computing and Engineering.* Kuching, Malaysia: IEEE.

Diaconu, R., Keller, J., & Valero, M. (2013). Manycraft: Scaling Minecraft to Millions. *Workshop on Network and Systems Support for Games* (pp. 1-6). Denver: IEEE.

Kwame, A. E., Martey, E. M., & Chris, A. G. (2017). *Qualitative Assesment of Comiled, Interpreted and Hybrid Programming Languages.* Ney York: Foundation of Computer Science.

Prechelt, L. (2000). *An Empirical Comparison of C, C++, Java, Perl, Python, Rexx and Tcl for a shearch/string-processing program.* IEEE Computer.

Xie, Y., & Aiken, A. (2005). Context- and path-sensitive memory leak detection. *ESEC/FSE-13* (pp. 115-125). New York: Association for Computing Machinery.