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. This class is called table and provides much of the functionality of the game, from controlling the spawning and removal of the stones and calculating the scores to managing the players and turns. This means that each of the sheets are treated as a separate game and so they are scored separately. The program uses a stone class to control the movement and interaction of the stones such as the collisions with the walls and the other stones. All of the drawn parts other than the stones are drawn from either the cushion array or the feature array. The feature array consists of the hog line, hack line and the rings forming the house. To store these in the same vector I used a parent class to store them in the vector and then dynamically cast them to the correct class when trying to draw the feature.

A picture containing diagram

Description automatically generated

Figure - Game running with 5 sheets

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.

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.

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.

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.

# Bibliography

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.

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.