# C++ CW 4 Documentation File

Name: Yutian Chen

Student ID: 20028267

**1. Introduction:**

This game is a rhythm game which simulates the **Triko Drum Master** game by enabling user to hit the drum with the keys on the keyboard. The **beat** is set to be 125ms.

There are 4 tracks corresponding to 4 controllers. They are controlled by **up, down, left, right** correspondingly. The controller can make voice.

**2. Compilation and running:**

Nothing special to consider.

**3. File list:**

|  |  |
| --- | --- |
| Code .h/.cpp Filename | What this file is for, e.g. what class is in it and what it does |
| states director | Store all the states’ information of the game.  The **State.h** is the common base class of the state above, and the **State\_funcion.h** is the common base class of **State\_start, State\_pause and State\_end** inheriting the **State** class |
| Button director | Store all the button components carrying different jumping/function logics.  The common base class is **Button** class which define the common function like **appearance**, and **click events** handling. |
| scripts director | The script class encapsulates the **save/load** logic of the system. |
| notes director | Define the note elements of the game.  The common base class is note.h/cpp |
| Ranking director | Store the ranking .txt file as well as the file reader. Ranking class is responsible for handling ranking logic. |
| controller.h/.cpp | Define the user controller which handle the player’s inputs.  Have the animation effects simulation the action of press |
| lifeManager.h/.cpp | Define the life elements in the game. |
| Avenue.h/.cpp | Simulates the avenue of tracks, when pressing ‘c’, the colour of the arrow will change. |
| luckyDraw.h/.cpp | Simulates the lucky draw box using the **tile management** and **image rotating.** |
| Avenue.h/.cpp | Add special effects for the tracks. |
| Configuration.h/.cpp | Define the game setting. |
| Speed controller.h/.cpp | Control the speed of notes. The element is draggable within a short distance of a bar. |

**4. Resource file list:**

I use the following images, which are all in the **image** director

|  |  |  |  |
| --- | --- | --- | --- |
| a.png | b.png | c.png | d.png |
| e.png | life.png |  |  |
| cherry.png | s\_cherry.png | b\_ cherry.png |  |
| watermelon.png | s\_ watermelon.png | b\_ watermelon.png |  |
| kiwi.png | s\_kiwi.png | b\_kiwi.png |  |
| lemon.png | s\_ lemon.png | b\_ lemon.png |  |

I also use the following audio clips, which are all in the “sounds” director

|  |  |  |  |
| --- | --- | --- | --- |
| bmboledg.wav | HH open.wav | PlasticTray\_PS.wav | Side Stick\_Kalavrezos\_2011.wav |

I also use the following .txt file, which are in the “scripts” director & “Ranking” director:

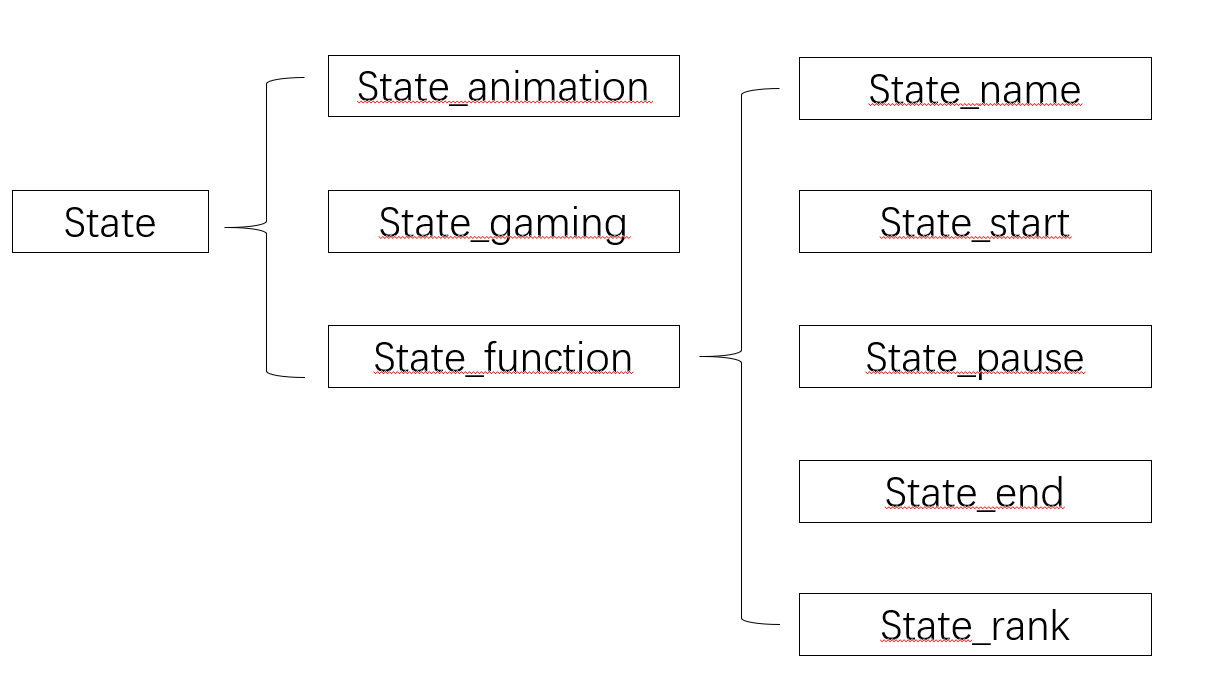
|  |  |
| --- | --- |
| beackup.txt | one.txt |
| Rank.txt |  |

**5. Requirements**

**Requirement 1: Add states to your program**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
| Gaming state | 00:38 | 00:50 | State\_gaming.h/.cpp  Scyyc1Engine.h (line: 24-42) the constructor initializes all the stage  Scyyc1Engine.h (line: 80) switchState() function  Handle the jumping from one page to another, others are the same. |
| Start menu | 00:31 | 00:37 | State\_start.h/.cpp  Scyyc1Engine.h (line: 80) switchState() function |
| Pause page | 1:22  1:30 | 1:23  1:33 | State\_pause.h/.cpp  Scyyc1Engine.h (line: 80) switchState() function |
| End page | 00:49 | 00:56 | State\_end.h/.cpp  Scyyc1Engine.h (line:19) Scyyc1Engine construcot  Scyyc1Engine.h (line: 80) switchState() function |
| Starting animation | 0:18 | 0:26 | State\_animation.h/.cpp  Scyyc1Engine.h (line: 80) switchState() function |
| Name reading state | 00:27 | 00:30 | State\_name.h/.cpp  Scyyc1Engine.h (line: 80) switchState() function |
| Rank state | 00:32  00:57 | 00:35  00:60 | State\_rank.h/.cpp  Scyyc1Engine.h (line: 80) switchState() function |

The basis structure of the states is following:



1. The **State** class is the common basis class which defines all the necessary common function of the pages. The **State\_function** is the intermediate class which defines the common functions for the menu-like pages (start menu, pause page and end page as well as name-reading page and ranking page).
2. The switching and jumping are handled by the **Scyyc1Engine** which passes a pointer of **stateId** element to the state below and the states with their corresponding components will change the stated in the engine and jump. The states are maintained in **a list of State pointer**. When jumping the, it will switch within the list.
3. The jumping action is handled by the virtMainLoopPostUpdate() function at the end of each loop. When the stateId by state class, the use of virtMainLoopPostUpdate() will clean up the object buffer for current state and call the switchState() function to switch and initialize.
4. The default parameters of the state including 3 elements, **name, engine, and stateId pointer.**
5. The creation and deletion are handled by switchState() function (line: 70 - 82 in Scyyc1Engine), which will take the number as parameter indicating which state to jump and dynamically create new state and pass necessary parameter as well as initialize it. The change of **stateId** variable is handled by the virtDoUpdate() function of “button”.

**Requirement 2: Save and load some non-trivial data**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
| Read in script | 1:45 | 1:48 | script.h/.cpp, one.txt and backup.txt  State\_gaming.cpp (line: 27-29) constructor  Create a script instance, passing the script name. And call buildNoteSet() function to obtain a note set.  State\_gaming.cpp (line: 301) call the getScritStream() function of the script class to obtain the **copy** of the notes script. |
| Output the half-done script to the file | 1:33 |  | State\_gaming.cpp (line 201-204)  When key s on the keyboard is pressed, calls saveScript() function to store the unfinish script to the backup.txt and switch to pause state |
| Check and add new score in ascending order | 00:57 | 00:60 | Ranking.h/.cpp, State\_rank.h/.cpp  State\_end.cpp (line 69-71) calls  addToRank() function of Ranking class to add and sort the new score to the class.  Save() function will save the new list to the file in order |

1. The structure of the .txt is following: (in script director)

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **When to appear** | **How long** | **Track** |
| Single | 2 | 0 | 2 |
| Long | 4 | 200 | 2 |
| Double | 4 | 0 | 1 |
| Single | 6 | 0 | 0 |
| Single | 2 | 0 | 2 |
| Long | 2 | 150 | 3 |
| … | … | … | … |

**Type**: what type of note is going to display.

**When to appear:** specify the time in terms of beats where the note is going to display. The beats is set default to 125ms in the configuration.h/.cpp.

**How long**: specify the time of the note lasting. Single and Double notes are set to 0 in default.

**Track:** specify which track the note is going to display.

Script:

1. The script.h/.cpp handle the detail loading and writing logic. The gaming state will new a script instance and load the **NoteSet** of script instance and the note will dynamically appear in the gaming process corresponding to the time.
2. The script file will store the above elements in vector separately. A vector<note\*> will be created and getScriptStream() function will return a copy of the note list.
3. When the player pauses and chooses “load & exit” at the pause state, the system will automatically store where he has finished in the backup.txt file with the same structure. When the user chooses “continue”, the system will load a **new script and state** about the backup file which store the progress last time and restart.
4. I have design a sample script and more scripts can be added.

|  |  |
| --- | --- |
| **Name** | **Score** |
| YutianChen | 1000 |
| Hahaha | 860 |
| default | 450 |
| jimmy | 360 |
| Default | 0 |

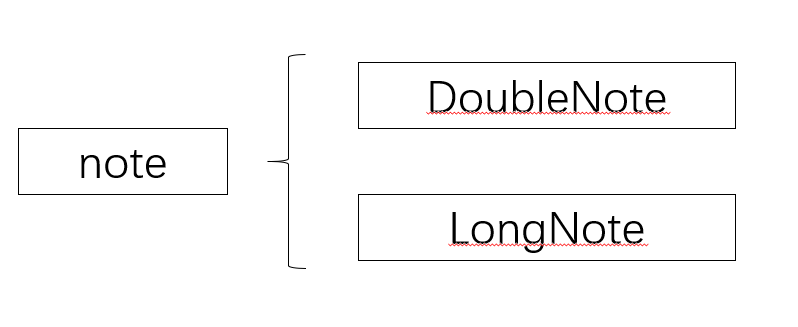
Ranking:

1. The ranking logic is handled by Ranking class including save() and load() function. Function addToRank() will take the name & score of current player and sort it to the list maintained in the class.
2. The State\_rank class is responsible for display the rank elements including user name and score and The State\_end will call addToRank() and save() to update the ranking when it is about to terminate.
3. The ranking only store top 5 item. Player without entering their name will be record as “default”.

**Requirement 3. Interesting and impressive automated objects**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |

|  |  |  |  |
| --- | --- | --- | --- |
| Note | 00:39 | 00:48 | note.h/.cpp, LongNote.h/.cpp and DoubleNote.h/.cpp  defines the appearances and move strategies.  State\_gaming() (line 252-278) virtMainLoopDoBeforeUpdate() generates notes according to the script with time.  State\_gaming() (line 361-373)  checkMiss() function handle the interaction with “life” tile manager. |
| controller | 00:32 | 00:50 | Controller.h/.cpp  defines the appearances and move strategies.  State\_gaming() (line 320-330)  The checkInput() function is called by virtKeyPress() function to deliver sound and handle the delection of notes that being caught. |
| Moving strategy | 00:32 | 00:50 | Note.cpp (line 49)  virtDoupdate() function add the speed (static variable for all notes) to the position. |



The appearance of notes & controllers: (watermelon, cherry, kiwi and lemon)

Single notes: 

Controllers: 

1. The notes in the four tracks will go from left to right according and **dynamically generated** according to the script. When the notes move close to the controller, pressing **UP, DOWN, LEFT, RIGHT** correspondingly. The controller will slightly “move down” which simulates the effect of being “pressed” meanwhile, making a sound of “drum”. There will be an evaluation for the performance of each note displayed at the dialog box in the down left.
2. The notes will be eliminated if the right controller is pressed and this is calculated by distance. There are three kinds of notes (long, double and single) with different appearance and calculation method.
3. Each evaluation will lead to different points. The logic is handled by EvaluationDetermination(int noteX, int controllerX) function (State\_gaming line 332). Each evaluation of “miss” will cause the decrease of “life” element, which is showed on the top right handled by tilemanager.
4. The dynamic generation is handled by virtMainLoopDoBeforeUpdate() function which calculates the number of “beats” from the former note. The detail will be explained in requirement D.

**Requirement 4. Impact/Impression (and requirement L: Sellable quality)**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
| Lucky draw box | 00:50 | 00:55 | Luckydraw.h/.cpp  In requirement C |
| Speed controller | 1:14 | 1:21 | speedController.cpp  State\_gaming (line 236) virtMouseMoved(int iX, int iY) method handles the modification logic of speed change. |
| Avenue |  |  | Avenue.h/.cpp  State\_gaming (line 198) call changeColou() method to change the colour of arrow (tiles). |

1. This game is a rhythm game which simulates the **Triko Drum Master** game by enabling user to hit the drum with the keys on the keyboard. The **beat** is set to be 125ms.
2. **Speed controller**: a static speed is maintained for all notes. There is a speed controller is the gaming state with a bar. Player can drag the speed controller along the bar to adjust the speed of notes. They can decide the speed they want for practice.
3. **Lucky draw box**: requirement C
4. **Avenue**: when player press c on the key board, the track arrows will change their colour when the notes pass through. (I forget to put this in the video but it works)
5. The game enables user to create their own scripts and backup when the player wants to quit. The system will remember where they are and store in a **.txt** file for player to load the for the former time and play again.

**Requirement A: Correctly implement scrolling and zooming using the framework’s FilterPoints class**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
| Filter point initialization | 2:31 | 2:34 | Scyyc1Engine.h  (line 19) initialization |
| zooming | 2:31 | 2:34 | Scyyc1Engine.h  (line: 60-66) mouse wheel event |
| scrolling | 2:31 | 2:34 | Scyyc1Engine.cpp  (line 41-46) update strategy for scrolling |

I have add scrolling and zooming effect for all the states. This enables player to adjust the game with the size the are comfortable. The scrolling is set to 5 pixel each (Scyyc1Engine.cpp, line 41-46).

**Requirement B: Have advanced animation for background and moving objects**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
| Start menu background | 00:31 | 00:37 | State\_start.cpp  (line: 7-14) In the virtBackgroundSetup() (line: 7-14) function creates the “snow” with different size  (line: 42-48) Then call the copyFrombackground() to produce a falling effect with each column. |
| Starting animation | 0:18 | 0:26 | State\_animation.h/.cpp (line 10-18) Use the fillBackground with different colour to update the background. |
| Button change colour | 00:30 |  | Button.cpp  (line 11-13) draw the button with different situation (mouse continuous holding or not)  State\_functional (line 27-52)  Handled the mouse clink event (mouse up & down) for button and maintain a button list |
| Rotating image | 00:50 | 00:55 | Scyyc1Engine.h (line 19) constructor initializes the totator.  Scyyc1Engine.cpp (line 66-91) handles the rotation update strategy.  State\_end.cpp (line 50-53) rotating the image with angle and display. |

1. The **start menu background**: I use the *copyFromBackground ()* function used in starfield demo to simulate the “falling snow” effect. Each piece of snows is in different size and falling in different speed. This is achieved by updating each column with different offset.
2. The pressable controller: When user press the key, the controller will go down a little bit, which simulate the effect of “being pressed”. Detail in requirement 3.
3. The **lucky draw box**: I implement a rotating kiwi image beside the draw box. Hopefully it can add the interest level of the lucky draw box
4. The starting animation: There is a starting animation with the background switching from black to white and go back to black again. (common opening animation) Achieved by fill background according to time
5. The button: when clicking the buttons, they will automatically change their colour. The base class Button define all the logic for pressing and the subclass inherits them and implements the update strategies.
6. I spent a lot of time analyse the evaluation system which aims to provide better experience for players. I also spent a lot of time to make a good appearance of both notes and controllers and make images fix to the size.

**Requirement C: Interesting and impressive tile manager usage**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
| lucky draw box set up | 00:50 | 00:55 | Luckydraw.h/.cpp  State\_end.cpp (line: 8-11) set up |
| Update strategy before being stopped | 00:50 | 0052 | State\_end.cpp (line: 56-60)  Increase the value to display the next picture. |
| Display result | 00:52 | 00:55 | Luckydraw.cpp (line 12) display picture according to the value of flag. |

1. The **lucky draw box**: I make use of **tile manager** and images shifting to simulate the lucky draw box in the end page. Before play clicking **“stop!”** thebox will keep animationg the “?” picture until being stopped. The box will then display one of the four fruits in the game into each small flag (these are done by changing the value in luck draw box). The result of the draw box will add to the final score of the player and each round of game only one chance (define in the configuration file).
2. The animation of “?” picture is handled by looping 5 parts of picture to simulate the effect of animation. The tilemanager maintains a list of images and the ending state switch between them by setting the value of the three flag.
3. The extra score criteria:

No same: 10 points

2 of them are the same: 50 points

All of them are the same: 100 points

**Requirement D. Creating new displayable objects during the game**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
| Notes (single, long and double) creation | 00:39 | 00:49 | Note.h/.cpp, LongNote.h/.cpp and Double.h/.cpp  State\_gaming.cpp (line: 252-292)  virtDoBeforeUpdate() function add notes to displayable object list. |
| Deletion | 00:39 | 00:49 | (line 173) virtKeyDown(int iKeyCode) function deletes the note for corresponding path if correct key is pressed.  (line 281-286) virtMainLoopDoAfterUpdate()function call checkMiss() function to delete the notes if they are out of bound. |

1. **Creation**: (handled by virtDoBeforeUpdate() function)

The notes are created in terms of beats. The time for appearance is recorded in the script (calculate the beats form the last note). I have set the **alignTime** variable which is used to define the updating time of creation specifically. Each time after a creation, the alignTime will be set to the same as the **current time adding beats of next note (line 258, State\_gaming)**. Only when the next time greater than alignTime will the system keep creating the next note (line 254). This ensure the in the same period only 1 creation will be conduct.

1. **Deletion:** The notes will be deleted in 2 situtations: (handled by virtDoAfterUpdate() function)
   1. When the notes reach the border of the right of the window
   2. When player press the corresponding key of controller.

**Requirement E. Allow user to enter text which appears on the graphical display**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
| Name reader | 00:26 | 00:31 | State\_name.h/.cpp  State\_name.cpp (line 36-38) bool values to indicate whether the player enter a capitalized or lowercase letter or numbers  (line 39) bool value to indicate whether the input string size is greater than 0 |

1. I have implemented a name reader with a individual state which enables user to enter their name for ranking. This is the second state.
2. If the player doesn’t enter anything, the system will use the name “default”.
3. Only characters & numbers are available (no space) and the max number is 10.
4. The deletion is handled by key “backspace”. The core idea is to maintain a **global string** called **PLAYER\_NAME** and each time when player enter a character or number, automatically adds it to the string and display the string. When user press “backspace”, delete one from the back and display.

**Requirement F. Complex intelligence on an automated moving object**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
|  |  |  |  |

**Requirement G. Non-trivial pixel-perfect collision detection**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
|  |  |  |  |

**Requirement H. Image rotation/manipulation using the CoordinateMapping object**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
| Rotating image setup | 00:50 | 00:55 | Scyyc1Engine.h (line 19) constructor initializes the totator. |
| Updating logic | 00:50 | 00:55 | Scyyc1Engine.cpp (line 66-91) handles the rotation update strategy. |
| Update angle and display | 00:50 | 00:55 | State\_end.cpp (line 50-53) rotating the image with angle and display. |

1. I add a rotating icon beside the luck draw box. Hopefully, it can make the process of “luck draw” funny. This icon is the icon of kiwi, which is my favourite fruit.

**Requirement I. Integrate sound using SDL**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
| Drum sound | 00:39 | 00:49 | State\_gaming.cpp (line:324)  Play the sound effect  State\_gaming.h (line:29-34)  Call the SDL\_mixer to initialize in the constructor  And display when user press keys |

I make use of **SDL\_mixer** framework and I have added it to environment setting.

When pressing the controller, it will deliver the sound of “drum”.

The sound files are stored in “sounds” director. The sound source is downloaded specific website from internet.

**Requirement J. Show your understanding of templates and/or operator overloading**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
|  |  |  |  |

**Requirement K. Use your own smart pointers appropriately**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
|  |  |  |  |

**Requirement M. An advanced feature I didn’t think of but you had pre-approved**

|  |  |  |  |
| --- | --- | --- | --- |
| *Item* | *Video start time* | *Video end time* | *Source files and line numbers* |
|  |  |  |  |