Asteroids

By Anna Kuznietsova

**START SCREEN**

We were initially given two options, one of them was to create a start screen for the game in order to give user more time to make the right decision when to start the game and the other one was to add an extra feature to the asteroid, so when the user destroys them, they are going to split into smaller pieces thus making the gameplay more challenging for the player. My choice has fallen on the first option and so I have set down to work. My start screen is a text label that is shown to the user at the beginning of the game and allows user to take a look at the battlefield. User can get to playing by pressing the “A” key which gives user the control of the spaceship and starts the game.

In my honest opinion, making a start screen has caught my attention at first glance and the more I thought about it, the more it seemed to make sense. While making the game and testing it, I found that the fact that you can get killed by asteroids at the very beginning of the game can discourage user from playing the game. I have thought of various ways to prevent that from happening, but the best one seemed to be adding a start screen, as there is no spaceship on the screen until the player presses “A” key. Not only does the starts screen prevents user from dying in the first seconds of the game, but it also adds a more strategical nuance to the game as now the player is responsible for the correct spawning of the spaceship which can either give player an advantage if used correctly. In view of the foregoing, making the start screen for the game seemed to make most sense and bring more benefits for the user.

Initially I have come up with a plan that would help me along the way of working on this project. I needed to decide on how to make the start screen, when it should and shouldn’t be visible to the player as well as choose the correct was of spawning the spaceship. I came up with a short plan to help me increase the productivity and make the most out of the game.

* One of the first challenges that I came across, was the creation of the start screen as I had a few different ideas of how to make it. One of the ideas that made it to the final project was to use the text label from it and of course declaring it in the “asteroids.h” file.
* One of the first challenges that I came across when creating the label was to decide when it should be visible to the player and when is the right time to make it disappear. The best was which would benefit the user was to choose a button, which after being pressed would make the label disappear (in the case of this game, the “A” key was chosen, but it can be easily changed to any other). By giving the user the control, it gives him an opportunity to choose the best suitable moment for him to press the button and get the most out of the game.
* Last but not least, I had to make a spaceship spawn on the user command. I have decided to use the “A” key which was given in the hint to the question. I have to move a part of the code which spawned the ship from the start of the game where it was initially made to a new function which would hide the text label and spawn a spaceship.

I have started with creating a label. To do this I have added this code to void Asteroids::CreateGUI()

In the code below you can the way I added the text label, as well as make it visible for the player and aligned it in the centre. After adding the label, I have added the GUILabel to the GUIContainer.

mStartGameLabel = shared\_ptr<GUILabel>(new GUILabel("Press A to start"));

mStartGameLabel->SetHorizontalAlignment(GUIComponent::GUI\_HALIGN\_CENTER);

mStartGameLabel->SetVerticalAlignment(GUIComponent::GUI\_VALIGN\_MIDDLE);

mStartGameLabel->SetVisible(true);

shared\_ptr<GUIComponent> start\_game\_component

= static\_pointer\_cast<GUIComponent>(mStartGameLabel);

mGameDisplay->GetContainer()->AddComponent(start\_game\_component, GLVector2f(0.5f, 0.5f));

Also it was necessary to declare it in Asteroids.h in the same place where other labels are:

shared\_ptr<GUILabel> mStartGameLabel;

Next, I have done 2 things when “A” is pressed: set label to be invisible and summon a spaceship (in bold):

void Asteroids::OnKeyPressed(uchar key, int x, int y){

switch (key){

case ' ':

mSpaceship->Shoot();

break;

**case 'a':**

**mStartGameLabel->SetVisible(false);**

**mGameWorld->AddObject(CreateSpaceship());**

default:

break;

}

}

And the last step is to delete creation of the spaceship in the beginning. To do this one line from void Asteroids::Start() need to be removed:

mGameWorld->AddObject(CreateSpaceship());

Here it is the result:

//image

**HIGH SCORE TABLE**

Initially I had a choice between high score table and bonus – power up system and my choice fell on high score table. I have decided to work on high score table as I thought that it would make the game more challenging for the players as well as motivate them to try and become better at the game and challenge their friends. Not only this allows you to challenge your friends, you can track your progress as you are getting better at the game. When combined, all the facts mentioned above, may lead to a more long term players that will be more motivated to come back to the game.

One of the most important factors that lead to this choice was the fact that I wanted to improve my skill base as well as coding knowledge and practice more on the reading and writing the information to the file. The ability to work more with read and write functions will give me a better understanding of how they work and the correct ways to use them.

On first glance on the high score table, I have thought of an idea to set a limit to the high score table of 10 people and 5 characters per name to make the game even more challenging as players have to constantly become better at the game to be able to stay on top of the high score table. This is also a way to pay respect to the classical asteroids game that was run on the arcade machines and didn’t have enough memory to store more that a small number of people in the high score table.

The next logical step was to make a user friendly high score list, to be able to achieve that, I will come up with a sufficient way of reading and writing information to and from the file as well as being able to sort and show the final high score table, limited to the 10 places to the player.

* One of the most important steps is to be able to correctly add information to the file and read from it in order for the information to be saved even after the game closes and of course it is the key to creating the high score table as it needs the player name and the score that he achieved.
* Another important thing that I have to figure out how to be able to transform the information from the file into the form of the list to be able to print it out and sort it in order for the player to be able to see the score in the decreasing order so that the player will be able to first see the top score and try and beat it.
* It is also important to be able to create a label for each of the players that are in the high score table so that it will look nicer and most importantly be shown to the user in the correct form. After the game is over, one of the most important issues is to figure out the correct way of cleaning the screen which includes deleting labels for lives, score, deleting the spaceship and asteroids and start displaying the high score table.

The code presented below shows how writing to file is done in the coursework, the function takes in a file name (string “filename”) as a parameter of the function and later the new document with that name is getting created and the information is written to the file. Program checks if there is a document with that name and if so, it adds information to the file and if not, it creates a new one. I used “ios\_base::app” in order to make a new information put after the latest one.

void Asteroids::WriteToFile(string filename)

{

ofstream MyFile(filename, ios\_base::app);

MyFile << mScore << " " << name << "\n";

MyFile.close();

}

This bit of code shows a ReadFromFile function which goes through the file and reads all the information presented there. I have first made a variable which will be used to later output the text from the file “string myText”. I have later added the code that adds the information to the vector and sorted it in descending order so that the highest score will be output first. Later on I have added the output for the text from the file and made the function close the file.

void Asteroids::ReadFromFile(string filename)

{

string myText;

ifstream MyReadFile(filename);

while (getline(MyReadFile, myText)) {

scorelist.push\_back(myText);

sort(scorelist.begin(), scorelist.end());

reverse(scorelist.begin(), scorelist.end());

}

MyReadFile.close();

}

The chuck of code shown below is where the labels are created for the high score table. I have made the function add the label to the vector called “highScoreLabels”. I have later aligned the label to be centered vertically and horizontally as well as to make the label visible. Later in the code you can see how I have made an offset on the y axis (“yOffset”) to make the labels be positioned one under the other.

void Asteroids::CreateScoresLabel() {

float yOffset = 1.0f;

for (int i = 0; i < 10; i++) {

highScoreLabels.push\_back(shared\_ptr<GUILabel>(new GUILabel(scorelist[i])));

highScoreLabels[i]->SetHorizontalAlignment(GUIComponent::GUI\_HALIGN\_CENTER);

highScoreLabels[i]->SetVerticalAlignment(GUIComponent::GUI\_VALIGN\_MIDDLE);

highScoreLabels[i]->SetVisible(true);

shared\_ptr<GUIComponent> place\_component

= static\_pointer\_cast<GUIComponent>(highScoreLabels[i]);

mGameDisplay->GetContainer()->AddComponent(place\_component, GLVector2f(0.5f, yOffset));

yOffset -= 0.1f;

}

}

In the code shown below, you can see how I have added all the declaration to the “asteroids.h” file

void CreateScoresLabel();

void WriteToFile(string filename);

void ReadFromFile(string filename);

std::vector<string> scorelist;

vector<shared\_ptr <GUILabel>> highScoreLabels;

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**DEMO MODE**

I have initially decided to make a demo mode as soon as I saw the part 3 of the coursework. I will have to come up with an algorithm to make the demo mode more believable and seem like the other player is controlling it. One of the first ideas that came to my head was to try and make it similar to the radar where the spaceship will detect the closest asteroid I have also created an algorithm for the spaceship to behave more human like. The algorithm includes the spaceship finding the closest asteroid, calculates the angle that it need to turn by to directly face the asteroids. The angle is calculated by using the basic geometry. After the spaceship is looking directly at the asteroid. This function should be in a cycle until “A” key is not pressed, but a problem occurs at this point. When the algorithm is being ran, it requires too many resources and thus no commands can be entered at this point.

A picture containing photo, black, sitting, white

Description automatically generated

I have decided to take another route and I have made a new automated control which is much simpler, but is quite effective, the spaceship spins around and shoots at the asteroids which allows the ship to destroy all asteroids around.

A screenshot of a computer

Description automatically generated

**APPENDIX**

Sometimes asteroids might spawn at the same position as the position of the spaceship which will cause instant death and the game has to be re – run.