ASD ASSIGNMENT

Project - 3D game
using OpenGL and C++

Mentor – Jon Macey

3/27/2013

Part - 2

Implementation

NCCA, Bournemouth University
Authored by: Navpreet Kaur Pawar

CONTENTS

ACKNOWLEDGEMENT

I would like to thank my mentor Mr. Jon Macey for his valuable ngl graphics library and demo codes that gave a good platform to understand the OpenGL and animation software development techniques.

I would also like to thank our lab demonstrator Mathew for his continuous support and advice through the project.

INTRODUCTION

Aim of the Project

The aim of the project was to develop a simple 3D game which allows for the detection of object collisions in 3D space and user control of the game characters.

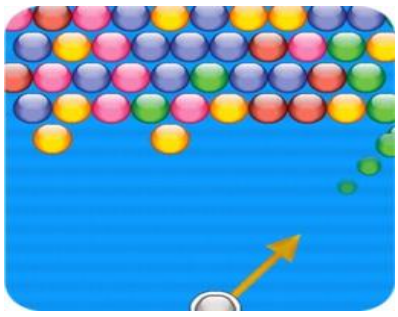
Keeping in mind the project requirements, the game was designed to implement Collision Detection algorithm using C++ and OpenGL.

Game Overview

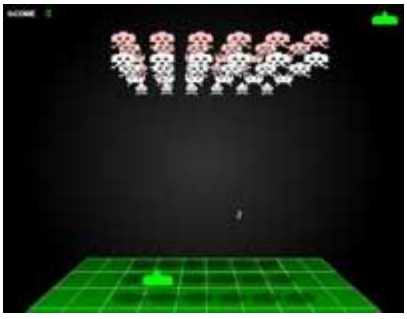
The Fruit and Insects game is setup in a cartoon forest environment. The player controls the basket using arrow keys on the keyboard to catch the falling characters. The objective of the game is to catch as many fruits in the basket as possible while avoiding other falling characters, namely the insects and the stones. Fruits give a positive score of 100 points for each fruit collected while an insect reduces your score by 50 points. The stones do not have any points. The basket has an initial shield value of 10. Reduction in shield value decreases player's life. The shield value is reduced by collision with stones. Each stone reduces basket's shield value by 1. The game ends if the basket's shield value is reduced to 0. The player wins the game if he catches all the fruits, while maintain his shield to last till the end.

Background Information

Related work - There are many games available where the objects are falling from a height and the player shoots/collects the falling objects. Bubble game and 3d space invaders are good examples. In both the games, the player shoots at the objects falling from a height and have their falling characters arranged in a grid like pattern.



Bubbles game



3D space invaders

The game project aims to have characters falling from height just like the related works, but at the same time arranging the falling characters randomly at a height. The player has to collect these objects, while avoiding objects with negative score, and ones that reduce its shield value.

The project followed the software SDLC and the following diagrams show the various stages of the project:

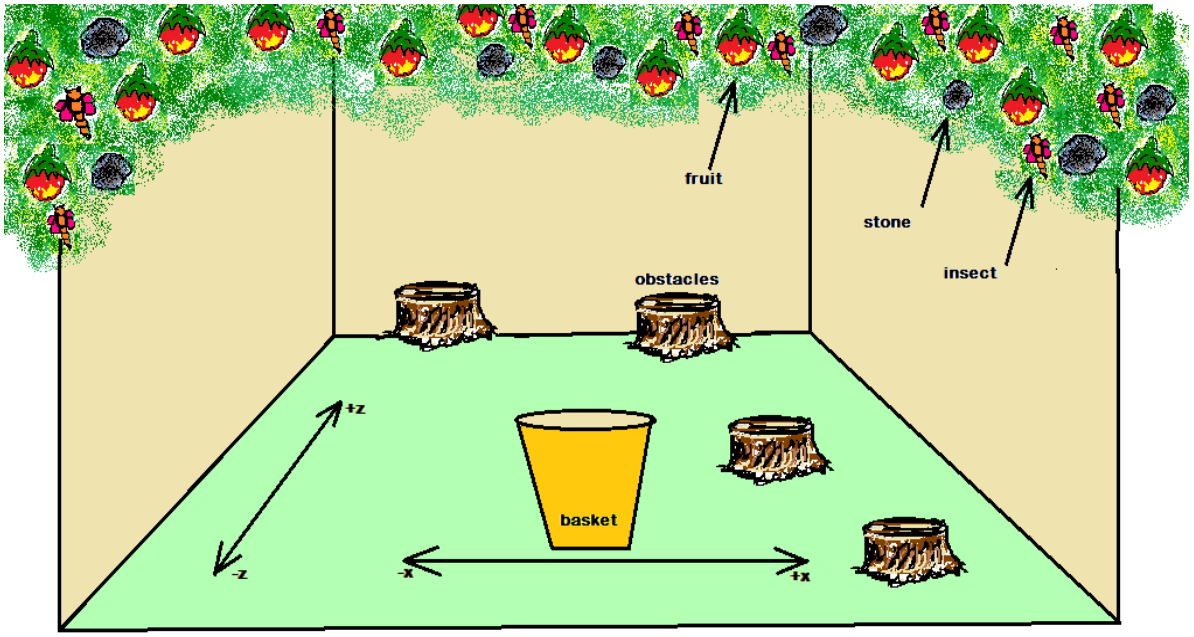


Fig -1: Initial diagram for the game scene

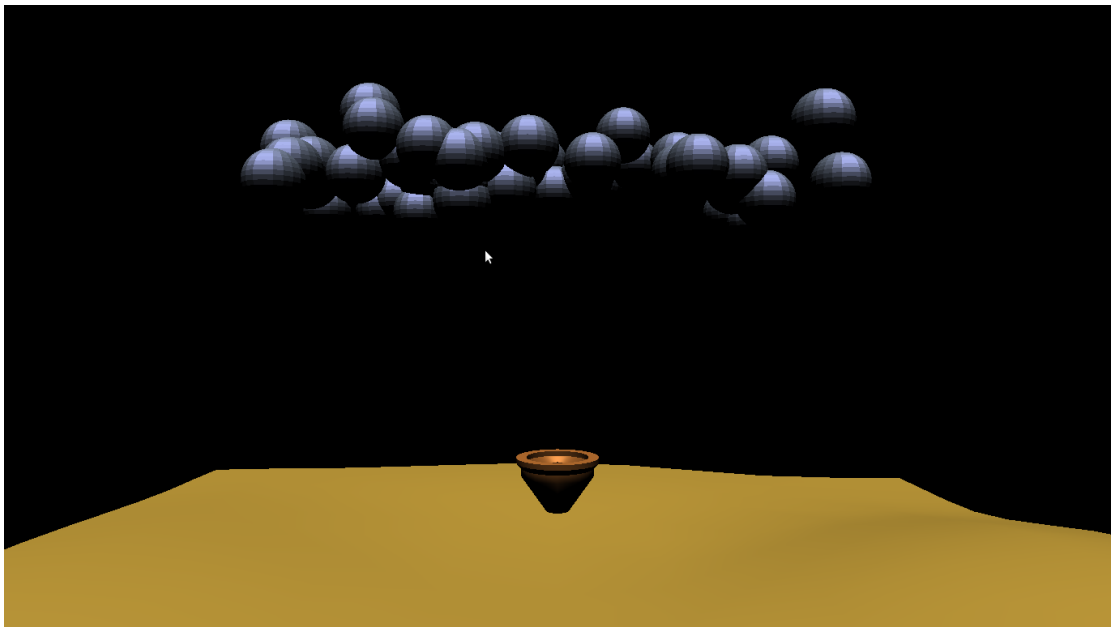


Fig -2: Screen shot of the prototype

Fig -3: Screen shot of the final game

ALGORITHMS AND TECHNIQUES

The game play is to catch a falling object. To determine this catch, the concept of collision detection is applied. Collision detection refers to the computational problem of detecting the intersection of two or more objects. In the project, the sphere-sphere collision algorithm is implemented to detect the collision between basket and the falling characters. We can say that if two objects positions are same, then the objects have collided. But it is not necessary for objects to have their centers collide. We have to take in consideration the object size or the radius. An object can collide if their surfaces touch. If the distance between the center of fruit and the center of basket is less than the sum of their radius, then a catch is valid.

Collision detection, in general, refers to the determination of intersection between two moving objects.

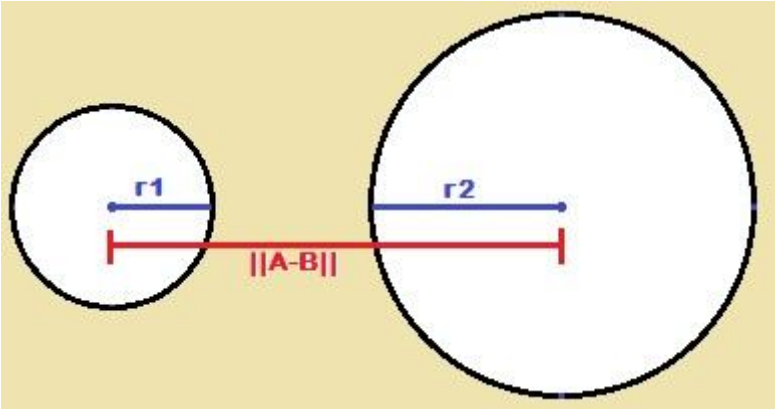


Fig 4 - Two circles *A* and *B* are defined as having a centre and a radius. They collide when the distance between the two centers is less than the sum of their radii:

$||A-B|| < (r1+r2)$

$||A-B||^{*2} < (r1+r2)^{*2}$

Amount of penetration, $p = ||A-B|| - (r1+r2)$

So, $||A-B|| - (r1+r2) < 0$

Normalized Vector, N = A-B) / ||A-B||, between the two centers.

Penetrating Vector, P = N*p

The two main parts of collision detection are –

- Detecting whether a collision has happened or not
- If yes, what should be the response to the collision?

In this project, each falling character will be assigned a radius that defines its bounding sphere. In every frame, falling characters will be compared against the basket, to find the distance between them. If the distance is less than their combined radius, then a collision is

recorded i.e. the basket has a valid catch. In case of a collision, the entities will respond to collision according to the behavior functions defined in the event of a collision. In case the collision is with a falling character, basket's response will be according to the type of falling character, which has been defined as an enumerated data type. This allows for conditional logic, depending on the type of entity that hits the basket.

- In case of a fruit, the basket's fruit count increases, score of the player increases.
- In case of an insect, the player's score is reduced.
- In case of a stone and a tree trunk, the shield value of the basket is decreased.

The collision detection applied in the game:

The basket and the falling characters are given a radius value, and are treated as spheres to detect collision.

Sphere 1: radius_1, position_1
Sphere 2: radius_2, position_2

Minimum distance for collision will be the sum of the radii of both the spheres.
Minimum distance = radius_1 + radius_2

Calculate the relative position for two spheres = position_1 – position_2.

Then calculate the relative length to compare with the minimum distance for collision, relative_length(position_1 – position_2)

Distance between spheres = [relative_length (position_1 – position_2)]$*^2$

Compare the minimum distance and the distance between spheres.

If distance between spheres < (minimum distance)$*^2$: - collision

Else: - no collision

PROGRAMMING

OpenGL – OpenGL stands for Open Graphics Library. It is the graphics API (Application Programming Interface) which provides the programmer an interface to the graphics hardware. The main advantage of OpenGL is its compatibility with multiple platforms like Windows, Linux, Mac OSX and portable devices. It is and open standard i.e. many companies can contribute to its development (does not mean that it is an open source).

Qt Creator - is a cross-platform C++ IDE (integrated development environment), part of the Qt SDK. It includes:

- visual debugger
- integrated GUI layout
- forms designer

C++ - the objects= oriented programming language, used in the project.

NGL – NCCA graphics library authored by Jon Macey

GAME STRUCTURE

Game Design

The game follows the basic game loop structure of:

- process
- update
- render

This concept is applied in games, as the game does not stop if there is no input from the user.

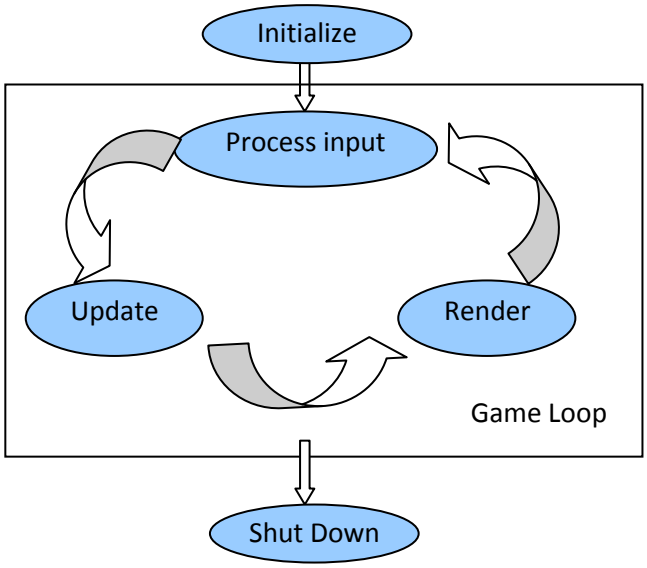


Fig -5: basic game loop

Process Input –The player's input is read, processed and translated into game action. To process the player input, the game needs to continuously check if there is an input from the user. The keys pressed by the user are stored in the event queue, which are translated one by one to appropriate action.

Update – the game is updated in the game manager class, and the transformations for various objects are updated. It is done independent of the user input and the rendering on the screen i.e. even if the user does not input, other objects will keep on moving around, perform their routine. In our case, the objects of class falling characters will keep on falling, even if the user does not move the basket.

Render – The screen is redrawn according to the change in position, rotation, etc. of an object. The render class will read the data about the update, and translate it to render calls.

Keeping in mind the basic game structure, the classes were designed for the different game objects. The relationship between these classes can be analyzed via the following class diagram:

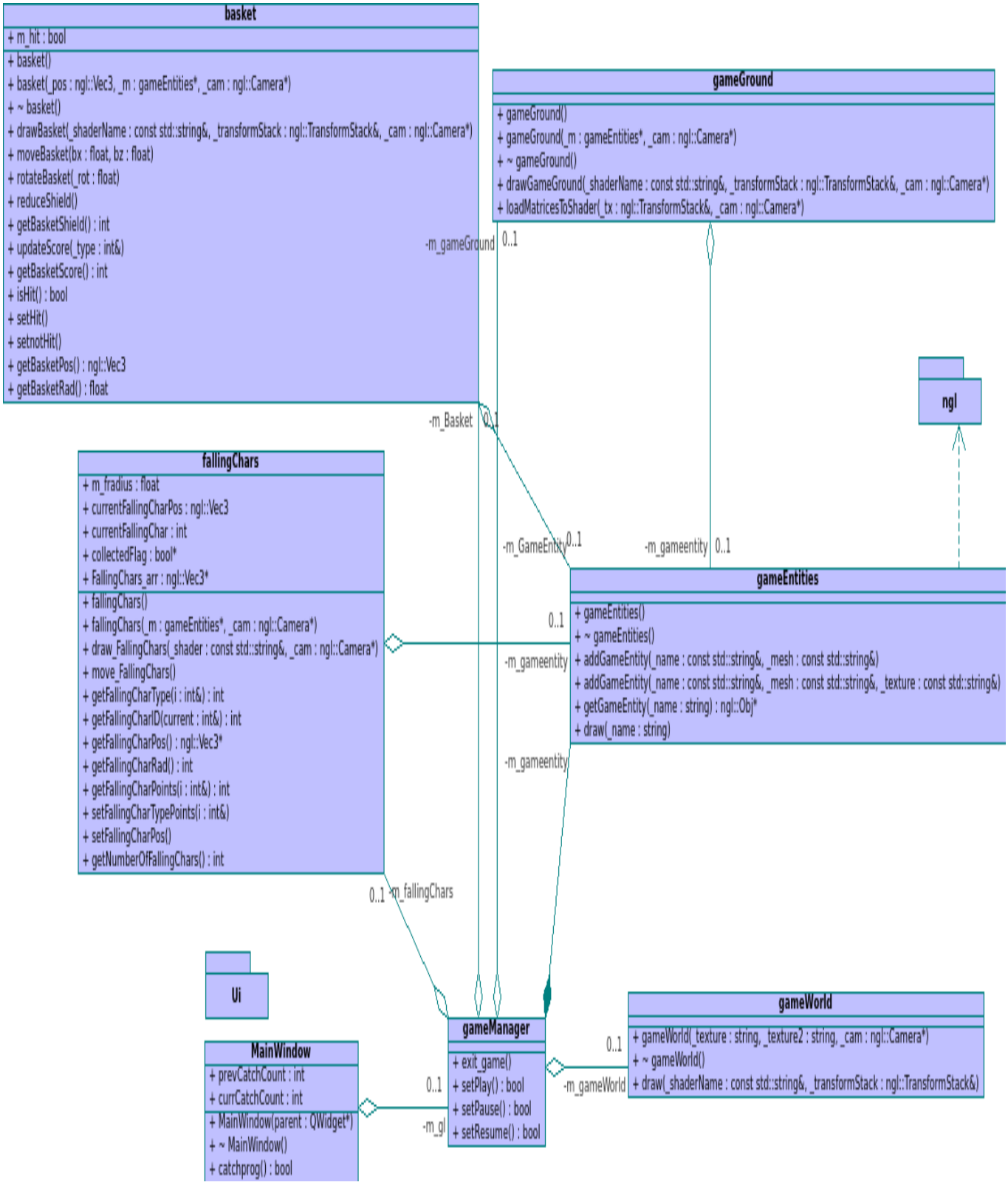


Fig 6 – Class Diagram for the game project

Main Classes and their Interaction

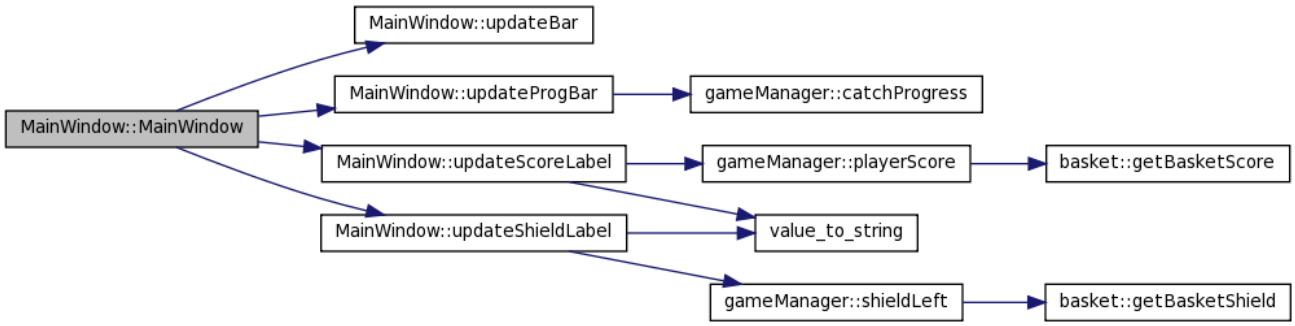
Basket (player) – the basket is responsible for catching the falling characters. The basket can move in the x-z plane on the game ground with its extents clamped to stay in the permissible area for game play.

FallingChars – this class defines the various objects that fall from a height in the game world. The falling character is identified by its id which is associated to the type, position, points of the falling character.

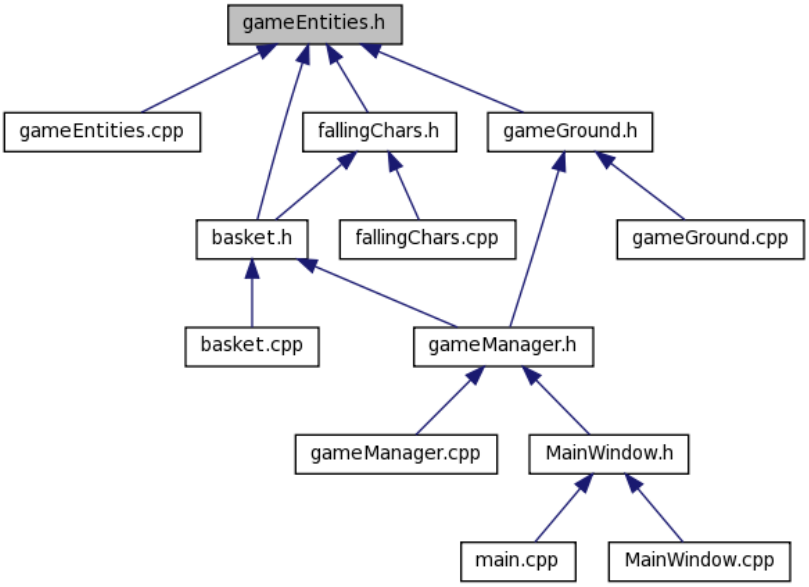
> *Falling Character Type* - The falling character type is defined as an enumerated data type. The falling characters are of three types, and the type of character is randomly assigned as fruit, insect, or stone, using ngl::Random class's randomPositiveNumber function between 1 and 3. The characters are falling from a height in the forest, which has been currently fixed at y = 25. Different points/shield values are associated with each falling character type.
>
> - For fruit , points = +100
> - for insects, points = -50
> - for stones, points = 0, shield= -1
>
> *Falling Characters Position* - the characters are assigned position randomly, using the ngl::Random class's getRandomVec3 function, keeping the y constant at 25 for all the characters.

Main Window – this class takes care of the user interface of the game. It creates the window for the game which can be resized. It stores the user input in the input queue and passes it to the game manager for translation.

It updates the progress bar depending on the catch progress of the user which is calculated by the game manager class. It even updates the player's score and shield in the user interface, with the input that comes from basket class regarding the basket state.

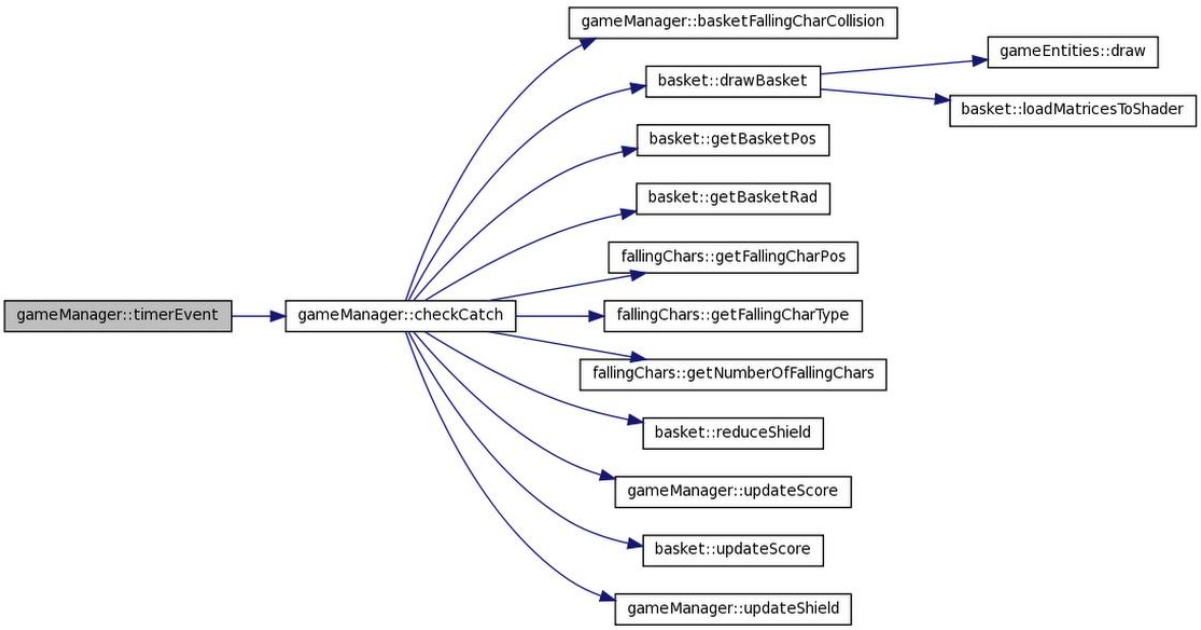


GameEntities - All the above classes – Basket, FallingCharcters, and GameWorld are drawn in a separate class called GameEntities. This class adds the various game entities into a map, with their meshes. The meshes can be accessed via a mapped key value. All the class pass control to this class from their draw function.

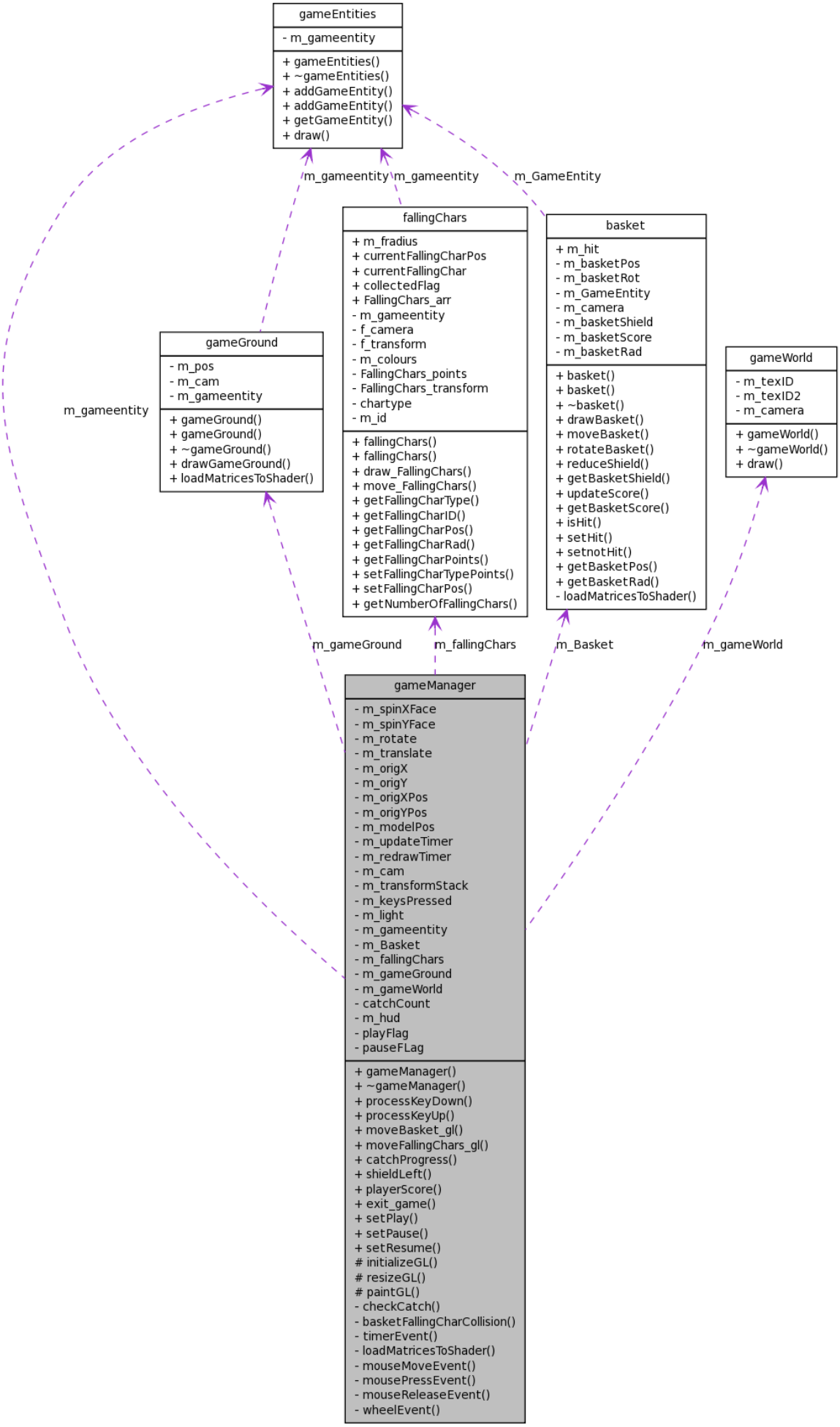


GameManager – this class performs all the major function of managing the entire game:

- Initialize – it initializes the screen, shaders, camera, light, game entities.
- Processes - the user input from the controller class i.e. the keys pressed, and translates it into transformation values for the game entities.
- Passes control to the game entities class to draw various objects.
- The game state of play, pause, resume and exit is handled by this class with the use of flags and qt signals and slots.
- The class takes advantage of the paintGL function (that is called to redraw the screen), to update the game object positions depending on the output of the move functions.
- Detects collision of game objects and updates the state of the player/basket in terms of score, number of catches and shield value. The main function to check for collisions – checkCatch() interacts with the basket and the falling character class to check for collision, and updates the game state depending on the collisions.
- Triggers time events, to check for collisions.



- The class even checks the game play win and loss. If the players shield is reduced to null, the player loses game and if player is able to catch all the fruits, then the player wins the game.
- It is the main class of the game , and its interaction with various classes of the game can be analyzed with the following diagram:


gameEntities
- m_gameentity
+ gameEntities()
+ ~gameEntities()
+ addGameEntity()
+ addGameEntity()
+ getGameEntity()
+ draw()
m_gameentity
m_gameentity
m_GameEntity
gameGround
- m_pos
- m_cam
- m_gameentity
+ gameGround()
+ gameGround()
+ ~gameGround()
+ drawGameGround()
+ loadMatricesToShader()
m_gameentity
fallingChars
+ m_fradius
+ currentFallingCharPos
+ currentFallingChar
+ collectedFlag
+ FallingChars_arr
- m_gameentity
- f_camera
- f_transform
- m_colours
- FallingChars_points
- FallingChars_transform
- chartype
- m_id
+ fallingChars()
+ fallingChars()
+ draw_FallingChars()
+ move_FallingChars()
+ getFallingCharType()
+ getFallingCharID()
+ getFallingCharPos()
+ getFallingCharRad()
+ getFallingCharPoints()
+ setFallingCharTypePoints()
+ setFallingCharPos()
+ getNumberOfFallingChars()
basket
+ m_hit
- m_basketPos
- m_basketRot
- m_GameEntity
- m_camera
- m_basketShield
- m_basketScore
- m_basketRad
+ basket()
+ basket()
+ ~basket()
+ drawBasket()
+ moveBasket()
+ rotateBasket()
+ reduceShield()
+ getBasketShield()
+ updateScore()
+ getBasketScore()
+ isHit()
+ setHit()
+ setnotHit()
+ getBasketPos()
+ getBasketRad()
- loadMatricesToShader()
gameWorld
- m_texID
- m_texID2
- m_camera
+ gameWorld()
+ ~gameWorld()
+ draw()
m_gameGround
m_fallingChars
m_Basket
m_gameWorld
gameManager
- m_spinXFace
- m_spinYFace
- m_rotate
- m_translate
- m_origX
- m_origY
- m_origXPos
- m_origYPos
- m_modelPos
- m_updateTimer
- m_redrawTimer
- m_cam
- m_transformStack
- m_keysPressed
- m_light
- m_gameentity
- m_Basket
- m_fallingChars
- m_gameGround
- m_gameWorld
- catchCount
- m_hud
- playFlag
- pauseFLag
+ gameManager()
+ ~gameManager()
+ processKeyDown()
+ processKeyUp()
+ moveBasket_gl()
+ moveFallingChars_gl()
+ catchProgress()
+ shieldLeft()
+ playerScore()
+ exit_game()
+ setPlay()
+ setPause()
+ setResume()
# initializeGL()
# resizeGL()
# paintGL()
- checkCatch()
- basketFallingCharCollision()
- timerEvent()
- loadMatricesToShader()
- mouseMoveEvent()
- mousePressEvent()
- mouseReleaseEvent()
- wheelEvent()

User Interface

Play button -sets the playFlag to true

Pause button -sets the playFlag to false

Resume button -setsthe pauseFlag to true

Progress Bar – updated to show the status of basket, how filled is the basket depending on the number of catches.

Score label – updates the player score

Shield label – gives the shield value left for the player

Exit button -setsthe playFlag to false and exits the

CONCLUSION

The game project is my first project with graphics and it helped me learn the animation software development techniques.

During the development of project, most of the initial goals have been achieved. The experimentation and implementation with the game has helped me in learning OpenGL and Qtcreater IDE , while brushing my C++ skills. The collision detection is one of the most challenging topics in gaming world, and the project gave me a platform to familiarize with the concept. I further wish to study the collision detection and reaction algorithms which were beyond the scope of this project, to help improve the game in future.

Further ideas:

- Multiplayer game functions
- A high score list to improve the user motivation
- Improving the collision detection

REFERENCES

- Jon Macey's class lecture notes and NGL demo codes

I went through a few blogs to understand the concept of game loop and game engines:
For collision detection-

- Chapter:20 Collision detection basics,2012. Available from - http://www.benareby.com/tutorial/ch20
- Firth P. 2011. Collision detection for dummies. Available from - http://www.wildbunny.co.uk/blog/2011/04/20/collision-detection-for-dummies/

For understanding the game loop:

- Willling W. 2009. Writing A Basic Game Engine, Available from - *http://www.ronkes.nl/blog/?2005-07-21-gameengine*
- Benstead L. with Astle D. and Hawkins K., 2009. Beginning OpenGL – game programming. Second Edition. USA: Course Technology, Cengage learning