****

Poo Poo Pipe

Technical Guide

Playtest date: Dec 14, 2019

GAM200F19KR

FALL 2019

**Name Title Coding Responsibility**

**Kim HaGyeong          Producer                   Architecture**

**Son JuHye     Graphic programmer   Graphics**

**Lee UiJin               Technical Director                   Architecture**

**Hong eunjin Designer                       Design, Sound**

# TECHNICAL OVERVIEW

Our engine is component-based engine that has 6 main systems, and they are implemented in one of the design patterns which is called singleton. Therefore, the overall engine could access to major system.

Major Systems:

* System (system.cpp)

Every component and objects which has a function initializing/updating, are implemented. By this implementation, every cpp and class updates and initializes at the same time.

* ObjectManager (ObjectManager.cpp)

It generates and destructs the objects by managing them.

* StageManager (StageManager.cpp)

It Adds levels and updates the change of the current level.

* Application (Application.cpp)

Checks the version of openGL and sets the size of the window. It keeps updating the events by the function PollEvents().

* Object(Object.cpp)
  + Makes the code easier to generate the components which is needed for the game and receives them.
* Component(FitPuzzleConponent.cpp)
  + Inherits component functions to mesh, transform, fitpuzzlecomponent and used for objects. The object is an element which is moveable. If the classes who needs these moveable objects, it is essential to use component.

Components

Our engine is object & component-based, therefore it will be able to combine the features (such as Transform, Mesh, FitPuzzleComponent etc.) from the basic object.

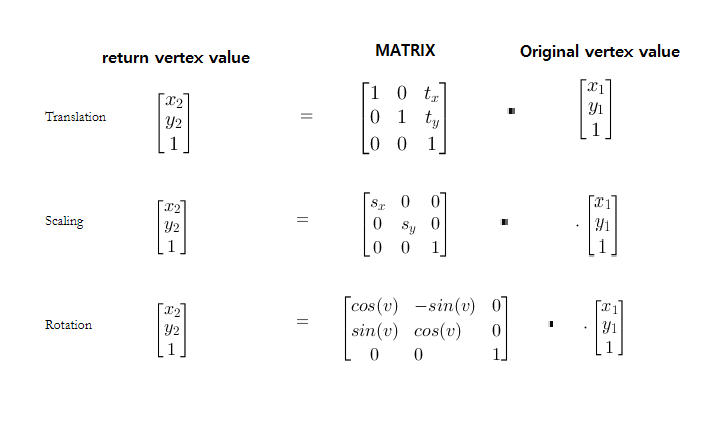
Component and object are major system in our game engine. Therefore, we can easily create basic object and add components to each object simply. After adding components to objects, all of them update in game runtime.

* Mesh

Makes variable shapes to object easily. It adds points to make variable shapes such as circle, ellipse, triangle, and rectangle. Moreover, it stores these points into Vertex. When rendering shapes, we choose each point which is needed and save them as vertices. Also, when loading textures, we choose and save the correct points as texture coordinates. In the Initialize function, vertices are set for each shape declared in the level as a Switch-Case form.

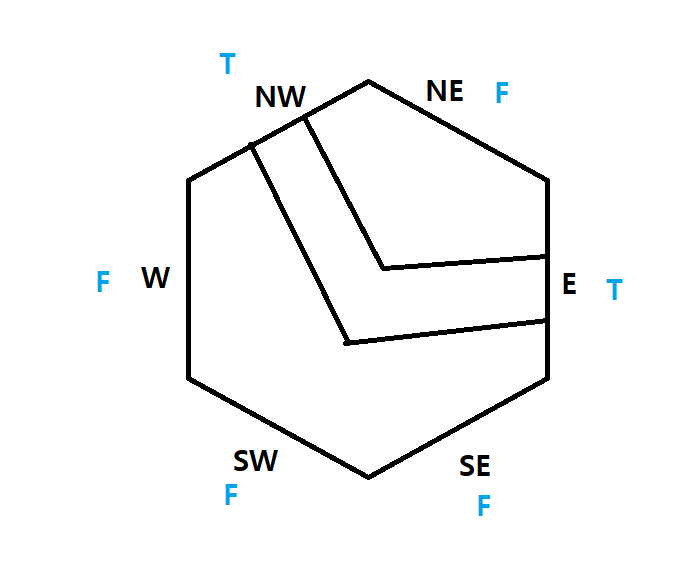
* Transform

Transform function store the value from translation, rotation, and scale. Then they return and set that value through get function and set function. In Level class, when calling the function translation, rotation, and scale in order, the variables changes to the state of each translation, rotation, scale function. Transform function receives the points by the mesh function, and returns the result of translation, rotation, scale which the points from the meshes are translated by each function.



* FitPuzzleComponent.cpp

This class is where it checks if the pipes are in connection. We made an array and send the information of the hexagon pipe. It uses enum with total 6 elements, ‘NW, NE, E, SE, SW, W’ which is the direction where the pipe could be connected. We saved them in bool so we distinguish them if the path of the pipes are opened or not. If it is opened it is saved as true, else, false. Every time the pipes rotate, these information inside array also update which rotates the location where it is opened. In Level class, we use the if function to check each object pipes are connected or not.



PuzzleComponent() : Component(COMPONENTTYPE\_PIPE)

{

dirArray1[0] = false;

dirArray1[1] = false;

dirArray1[2] = true;

dirArray1[3] = false;

dirArray1[4] = false;

dirArray1[5] = true;

}

void PuzzleComponent::Update()

{

bool temp = dirArray1[0];

for(int i = 0; i < 6; i++)

{

dirArray1[i] = dirArray1[i + 1];

if(i == 5)

{

dirArray1[i] = temp;

}

std::cout << dirArray1[i];

}

}

Major technologies

* Graphic API
  + OpenGL
* Graphic API library
  + Glfw
* Audio API
  + FMOD
* Custom Object
  + Component based custom engine

# CODING METHODS

Describe/show all coding conventions that are to be used on the project, including file naming conventions, file locations (i.e., which files go in which folders in the project), code formatting, and code documentation.

* Code formatting

1. The variables that are used to check the status in conditional statements are included at the end of the variable name.

Example) bool connectCheck = true;

Int moveCheck = 0;

1. Variables that are used repeatedly for each level has the same name but added in different number at the end.

Example) Object\* movePuzzle2;

Object\* movePuzzle2\_2;

1. Every name of variables is written by low camel case.

Example) bool connectCheck = true;

Int moveCheck = 0;

1. Every name of functions is not set by any rules. However, it is written by upper camel case in usual.

Example) void SetDirection(bool a, bool b, bool c, bool d, bool e, bool f);

Void SetObjectID(ObjectID objID);

1. Every name of cpp and header files is written by upper camel case.

Example) Object.cpp

ObjectManager.cpp

System.cpp

1. All name of filter(folder) is written by upper camel case and is related to cpp files which folder they are included.

Example) Application

Graphics

Component

1. All of name of classes is written by upper camel case.

Example) class Object

Class PuzzleComponent

Also describe/show what kind of source control system you will use and any rules your team has about its use.

- Team 99 uses *‘source tree’* for source control system. Repository of team99 is ‘*[git@bitbucket.org:hagyeong/new\_99.git](about:blank)’*.

- There are some branches in source control such as ‘architecture’, ‘graphic’, ‘audio’. We work and upload in each branch and after finish works, branches merge together and upload in master.

- When we upload the files into source tree, we describe about the things that changed or fixed previous file.

# GRAPHICS OVERVIEW

Show/describe the structure of the graphics engine.

Currently our graphic engine uses glfw to bring up the window. Each level has its own input functions. Because the graphics engine uses opengl API, it uses fragments and vertex shaders. The graphics engine uses a custom math library.

Make sure these questions are answered:

* What techniques will be used to implement the graphics design?

We will add font, animation, and camera functions to implement the graphic design. Fonts will be displayed using bitmap fonts.

* What graphics API will you use?
  + OpenGL
* Are you using fixed function or shaders?
  + Our graphic engine uses fragment shader, vertex shader.
* How are you loading assets such as sprites, models, textures, and animations?
  + To pop up the spirtes, models, textures, and animations, we are using stb library.

# ADDITIONAL TOPICS

* Art pipeline

There is no function that can change image in game runtime, however, image can be changed by inputting code in cpp.

Example)

* Audio pipeline

There is no function that can change audio in game runtime, however, audio can be changed by inputting code in cpp.

Controller is space bar that can stop audio in game runtime.

The background music sound is called by the extern variable of the main menu. By this single initialize, it creates the sound and plays continuously before the ‘Stop’ function is called. The stop function is called at start of the initialization function of the main menu. Therefore, when the game is cleared and gets back to the main menu again, it encounters the stop function and stops the sound, and does the re-initialization which resumes the background music from the start.

The sound effects are also called by the variable by each level. It is called the same way as how the background music is initialized, create, and play. It initializes and create sound by the Initialize function then plays the sound by the count of the rotation check. Therefore, the sound of the sound effects only plays when the player tries to rotate the puzzle pieces.

* Scripting languages: English