About the AGE Engine

The AGE engine is a simple 2D console game engine which runs in UNIX operating system. It provides a considerable amount of C++ APIs for clients to build and create a 2D console-based game that runs in UNIX.

Breakdown && Plan of Attack

The core architecture of the AGE engine is designed under the software design pattern named Model-View-Controller (MVC) that divides the functionalities of the engine into three main conncected components. That is been said,

First Step - Controller - DDL - 11.27 (one day)

First thing I need to do is to use an adapter pattern to wrap the API's from the third library NCurses in order to make it more friendly with C++. Secondly, implement two simple `Controller` classes to get the user input from the real-time curses and keyboard.

Moreover, to implement a fix rate of frame in the game engine, my idea is to encapsulate such a feature into the controller. Every time when `getAction()` invokes, say the controller is set to FPS with 60, the controller will only try to get user input for every 1/60 second. If there is an input we simply return it. Otherwise, we tells the model there is no user input, you simply refresh all the objects you have in there.

It probably takes some time to play around and build some helper functions / APIs for the view parts to use later on during the milestone.

Second Step - WindowModel - DDL - 11.28 ~ 11.29 (two days)

In my current design, there are two types of models: `WindowModel` and `ObjectModel`. First, I need to complete the `WindowModel` which is an abstract class that is able to contain other `WindowModel` as sub-windows. Then I need to complete two other windows: which are provided by the AGE engine. Two pre-designed window named `StatusWindow` and `BoardWindow` and they are combined into one big window named `GameWindow`. Since `WindowModel` has a good extensibility, clients can implement their own customized window and combine them into one complete game window as well.

Notice that `BoardWindow` owns an abstract class named `Grid`. One of the implementation is for enhancement and we will implement it at last. We first need to implement a `Grid` named `SimpleGrid` which is a simple 2D array buffer that stores the actual picture of the game.

Note that `WindowModel` contains an other class named `ObjectModel`.

Third Step - ObjectModel - DLL - 11.30 (one day)

AGE engine also provides types of object / entity creation for the clients. It is a data structure for handling object/entity logic and storing its data. As started, a simple abstract class named `ObjectModel` is needed. Then I need to implement the three concrete derived classes named `ASCIIObject`, `RectObject` and `BitmapObject` and relevant data types. All of them should be fairly easy since their is not much of logic to implement.

Forth Step - View - DLL - 12.01 (one day)

Before we continue to implement our ObjectModel system. To ensure we have done fairly correct so far, I need to implement a `view` component for the MVC first in order to see if my engine works so far (note that `WindowModel` contains references to `View`s). `View` class is an abstract class that are suppose to handle the logic of the actual displaying. Its functionality is the bridge between the data and the player. Since all the data are already handled correctly in the `Grid` class. A `view` and its hierarchy should be fairly simple and can be quickly done in one day.

Fifth Step - Motion && Gravity - DLL - 12.02 ~ 12.04 (three days) (adding one extra day for debugging)

One of the big chellenge is to handle the motion and graivity on each of the `ObjectModel`. First, I should use Decorator Pattern to create one of the decorator named `MovableObject` (the other one is `CollidableObject` which will be implemented on the next stage). This decorator has an extra field named `Velocity`. There is another class derived from `MovableObject` is named `GravitaionalObject`. The reason GravitaionalObject is not one of the decorator is because an object to have a gravity acts on it, it must be able to move first.

During these processes, I probably need one extra day to solve unexpected bug situation. Moreover, I might encounter some designing problems when we are actually implementing the motion && graivity system. I might also need some time to restructrue the partial system as well.

Sixth Step - Collision - DLL - 12.05 (two days)

To support different `ObjectModel` can collides with each other. We implement the second decorator on `ObjectModel` is called `CollidableObject`. Which will definitely using Visitor Pattern to implement such a feature. I need to implement each of the type with its corresponding collision logic for possible collision with other types.

Seventh Step - enhancement - Quadtree - DLL - 12.06 (one day)

Recall that in a typical 2D board with n entities on it. By using brute force, it requires a total n^2 numbers of detections for all of possible collisions in each frame. Which consumes an incredible amount of time usage. To improve such situation, My parnter and I decide to use a quadtree to decrease the time complexcity into $O(\log(n))$. Which means, it only takes a total $\log(n)$ numbers of detections for each frame. Which is way more faster than the brute force.

In the other hand, it takes quite a lot of memory. So it is an enhancement feature that up to clients to enable or disable it. The tree itself is not hard. As long as the previous class `SimpleGrid` is implemented well, this part should be fairly easy and can be finished in one day.

Small Trival Enhancement - Color / higher FPS / larger console - DLL - 12.07 (one day)

This day should be implementing some fairly simple enhancements that can be easily enabled and disabled.

Last Step - Game Number Two - Super Mario Bros. World 1-1 - DLL - 12.07 ~ 12.09 (three days)

One of the game I want to make is Super Mario. Instead of the whole game, I will only implement the first level of the game. Gravity, Collision, Board Types (view) and player controlled aspects can all be shown in this game. It would be a great game for me to show the power of this AGE engine. Moreover, in the enhancement version, the game can display color and with hight FPS, the game should be more playable than the normal version.

Last Last Step - Game Number Two - Conway's Game of Life - DLL - 12.10 (one day)

This one should be fairly easy. The main reason to choose this game is because this is a great oppotunity to show the power of quadtree (decrease the time usage during the edge detecions).