Tutorial – State Diagrams

Introduction:

In this tutorial (and in the tutorials for the remaining UML sessions) we are going to analyse a simple game so that we can document its design using UML diagrams.

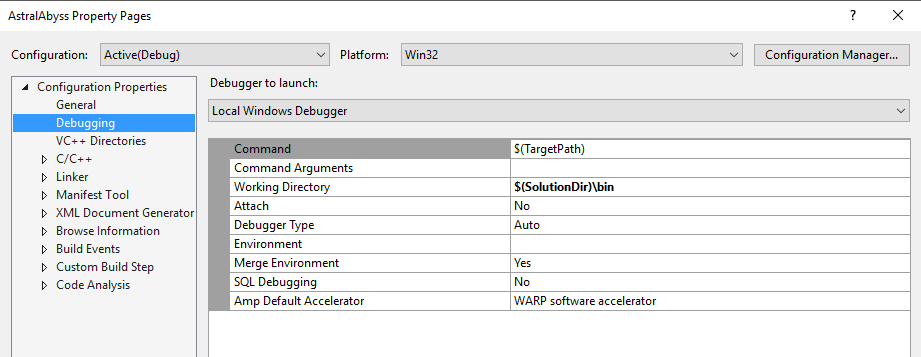
In this tutorial we will create a state diagram for the game Astral Abyss.

In the exercises for the session on *Class Diagrams* you would have created a class diagram for this game. Completing this exercise will assist you in this tutorial, so you may find it useful to complete that exercise if you have not yet done so.

Set Up:

The Astral Abyss project is available on the Resources page for this subject. If you have already added this project to your *aieBootstrap* solution, you can skip this step.

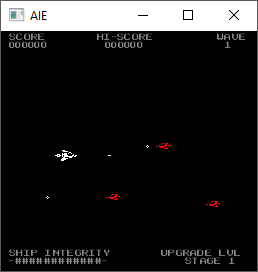
The game is provided as a project that will link into the *aieBootstrap* solution. If you do not yet have a copy of *aieBootstrap*, you will need to download that from this github repository: <https://github.com/AcademyOfInteractiveEntertainment/aieBootstrap>

1. Download *AstralAbyss.zip* from the *Resources* page for this subject
2. Extract the zip file to your computer. (A good place to extract it would be to the bootstrap solution folder)
3. Open the *aieBootstrap* solution
4. In Visual Studio, add the *AstralAbyss* project to the solution.  
   In the *Solution Explorer*, right-click on the solution and select *Add -> Existing Project*
5. Open the properties for the *AstralAbyss* project and ensure the debug *Working Directory* is set to **$(SolutionDir)bin\**  
   
6. Lastly, we need to copy the images and fonts this project uses to the solution’s *bin* folder.

In the AstralAbyss project folder you will find a *bin* folder containing images and fonts. Move these into the **$(SolutionDir)\bin** folder.   
  
If your game launches and you cannot see anything drawn, you have likely copied the resources to the wrong folder.

Once you have set up the project, set it as the active *Start Up Project*, compile and then run the project.

You should be able to launch and play the *Astral Abyss* game without errors.



Creating a State Diagram:

In a larger game you may find state machines being used to handle player or enemy AI logic.

Within the *Astral Abyss* game, only the *AstralAbyssApp* class makes use of a state machine. The functioning of this class is a logical choice to map using a state diagram.

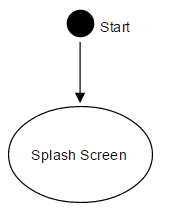
Take a look at the *AstralAbyssApp::update()* function. Here you will see that the game will select from one of several update functions to call depending on the value of the variable *m\_gameState*. This processing is replicated in the *draw()* function (so that each update function has a corresponding draw function).

The states listed here are the ones we will draw in our diagram. Specifically, these states are:

* Splash Screen
* Main Menu
* Game Play
* Pause Menu
* Game Over
* Score Board

We’ll start with the first state first, the *splash screen* state.

Draw the name of the state in a circle:



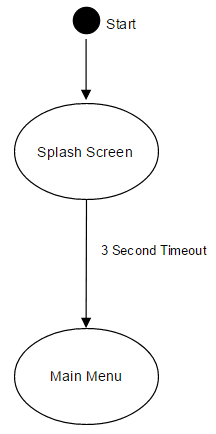
Because the game starts in the *Splash Screen* state, we add the *start* node at the top.

We need to determine the next state (or states) control of the program can flow to from the Splash Screen state, and what event (or events) causes control to transition to a new state.

If we take a look at the *updateSplashScreen()* function, you’ll see that after 3 seconds the *m\_gameState* value be set to the *Main Menu* state.

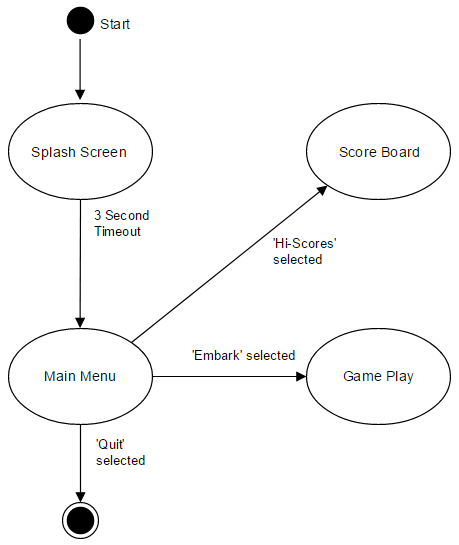
So the only other state that can be transitioned to from *Splash Screen* is *Main Menu*. And the event that causes this transition is a 3 second timeout.

We can add that information to the state diagram:



When you play the game, you’ll notice that there are three options available in the main menu: play, score board, and quit. You could have also determined this information by looking at the *updateMainMenu()* function.

The only way for the main menu state to transition to any of these states is for the player to select the option and press the space bar. Again, we can show this information in the state diagram:



The transition event for each of our new states is the selection of the corresponding menu command during the main menu state.

Because selecting ‘quit’ will terminate the program, we’ve used the *end* node instead of another state.

Exercise:

Our state diagram is not yet finished, but by now you should have a good feel for how the state diagram is created.

Each state will be a very distinct state in our application – you can generally tell when a new state is needed because input, processing and output will be different.

The transitions between states are events that cause this change in processing. Selecting menu options, timeout events, or things happening in the game world (like the player dying) are all events that could cause a state change.

Examine the source code in the *AstralAbyssApp* class and complete this state diagram.

As you make your diagram, ensure that all transitions (arrows) have labels that describe the event that causes the state change.

Also, make sure that the program can never get ‘stuck’ in a state. It should be possible to keep moving around the diagram until we get to the *end* node.