Hector Medina

Professor Berthiaume

GAM 377 User Improvements/ ease of use improvements.

* **Changes to the Input Manager/ overall input system**
  + Added a default KeyEvent (for both Key up and Key Down) when calling a registration for an input key.
    - Reasoning: As I was trying to register for a bunch of keys in order to test Adding time, subtracting time, and checking the current time of the alarm, I ended up having to write a lot of registration calls for different KeyEvents, some of which required a key up and key down event. I thought that if I had a Default state that included both states, it would save time in certain cases.
  + Added a function to allow an Inputable to be able to deregister from all of the keys that it had registered to with just the one function.
    - Function: Inputable::DeregisterForAllInputKeys(), which goes through the SceneManager->Scene->Inputmanager then calls InputManager::DeregisterAllInputKeys(Inputable \* in).
    - Reasoning: There were many times when I was testing a multitude of sections of my game engine and deregistering all of the keys became extremely frustrating and I thought it could have definitely been streamlined, especially since keeping track of registered keys is fairly easy.
  + Added a function that checks if a key is being pressed on the frame that it is called upon. (not sure if I added this quarter or not...)
    - Function: InputManager:: IsKeyPressed(AZUL\_KEY key)
    - Reasoning: It is great that there is an event based callback for when a Key is pressed and released, but there are countless times of when I would like to know if a key is pressed throughout several frames (such as a moving a GameObject in the Update() call). (This was apparently not required based on the Diagrams shown in the last quarter.)
* **Changes to GameObject Movement**
  + Instead of putting in a matrix to change the position, scale, and rotation of the GameObject, I replaced it with three Vects representing, the position, scale, and rotation.
    - Function change: From SetGameObjectWorld(Matrix& newWorld) to SetGameObjectWorld(Vect& pos, Vect& scale, Vect& rot).
    - Reasons
      * As I was setting the position of the Gameobject, I noticed that I kept having to create a Matrix from 3 different vectors : position, scale, and rotation, whenever I called SetGameObjectWorld(). Therefore, I came to the conclusion that creating this Matrix could be done under the hood, so my SetGameObjectWorld does this process.
      * The Sound System I added required setting the SoundConscious entities (a Listener in the Sound system) via a Vect representing position and rotation. Having the transformations passed in as Vects made it extremely easy to set the position of the SoundConscious as well as be a bit more efficient since I would not need to extract the individual transforms from a matrix passed in.
  + Added a conversion from degrees to Radian in the P2Math class.
    - Function: P2Math::DegToRad(float )
    - Reasoning: Due to the fact that I was born in the U.S. I am used to degrees and not necessarily Radians. Due to the rotations being in Radians, I wanted a simple function to make it convenient for people who do not know what 90\* in radians is or 86\*.
* **Changes to GraphicsObjectMaker**
  + Fixed a Bug of not having freed up the memory of a graphics object that was to be removed.
    - Thanks to Zac Gross's post on how I was not calling the removal of the Drawable's graphics object on destruction, it led me to realize that I was only removing the graphics object from the list and not from memory.
* Changes to Drawable:
  + Added a call to the destructor so that the Drawable's Graphics Object would be removed (fixed a bug).
* **Testing the Alarm system**
  + All of the functions that I had made seemed to be working in a way that was easy for the user to use.
    - Tested the following
      * adding Time to an Alarm
      * subtracting Time to an Alarm
      * getting the time remaining of an Alarm
      * canceling an alarm
  + Possible additions for the future
    - One thing I thought about doing, but did not end up having the time to implement because of the amount of time that the sound system required, was to add the capability to pause an alarm.
      * Reasoning: I figured that there could be situations in which a user may want to pause an alarm, such as pausing a countdown timer such as a bomb that will explode when a certain event happens.
* **Attempting to create a simple Camera Manager for use as a SoundListener.**
  + I succeeded in creating a simple camera manager, but it came out sloppy as I thought I could not modify the Azul Camera Manager.
  + After I was given the go to modify the Camera Manager, I added a function to allow the user to set the current Camera, which would then allow the User