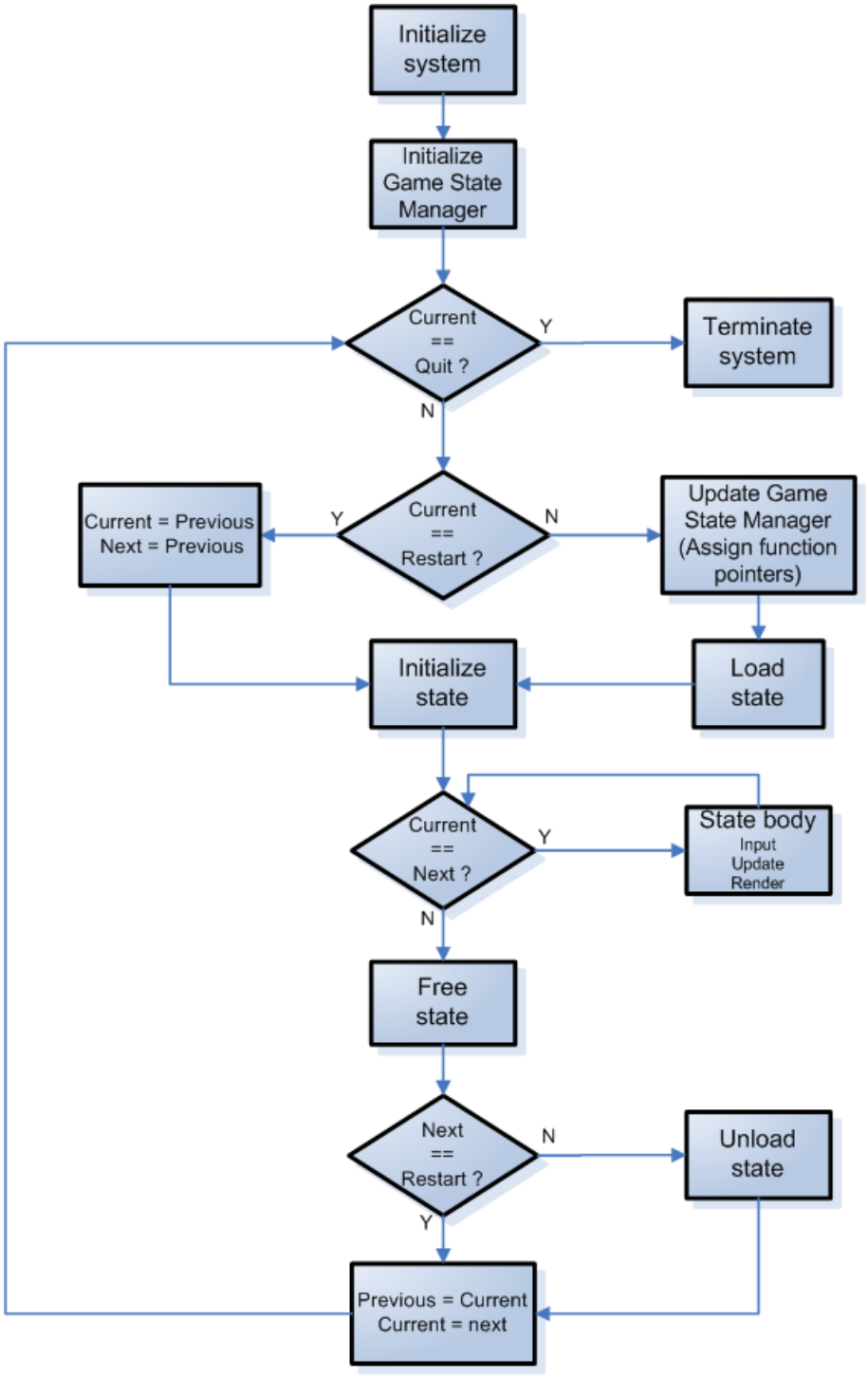
Lecture 3a

Game State Manager
Function Pointers

CS529
Fundamentals of
Game
Development

1. Game Engine Design

1.1. Game engine flow



- Any application consists of a general initialization part, followed by a loop that keeps running until the user decides to quit the application. This loop should handle all user input, handle how the user is interacting with the application, execute all game logic and finally reflect the overall result on the screen. In a game, this loop is called the "game loop". When the game loop breaks (The player decided to quit the game or any other possible reason), the application will go to a "freeing" part, which basically releases all the data that was loaded during the initialization part. Note that all data that was loaded inside the game loop should be released before exiting it. As a general rule, all data that was loaded in one component should be released in the same component upon exit, which will make the code more consistent, the application more stable and memory leaks much easier to trace.

- **Pseudo Code**

```
Initialize system components
Initialize frame rate controller
Initialize game state manager
While not quitting the game
    Call "Update()" game state manager
    Call "Reset()" frame controller
    If NOT restarting the current game state
        Call "Load()" current game state
    else
        Next game state = Previous game state
        Current game state = Previous game state
    Call "Init()" current game state
    While next game state is equal to current game state
        Call "Start()" frame controller
        Update input status
        Call "Update()" for the current game state
        Call "Render()" for the current game state
        Call frame controller end
    Call "Free()" for the current game state
    If NOT restarting current game state
        Call "Unload()" for current game state
    Previous game state = current game state
    Current game state = next game state
Terminate system components
```

- **Managing system components**
  Any game code (or application in general) has to initialize some system components before actually going into the game code and the game loop. These system components usually set up some necessary hardware related functionalities which will be later used within the game. Example:

  - Setting up a input device
  - Setting up video device
  - Setting up an audio device
  - Allocating video buffers
  - Deciding if some parts of the pipeline should be done using the hardware or software.

  This kind of system component initialization should be done just once before entering the game loop, and if any component fails to initialize properly, the application or the game usually quits with the appropriate error, since the application won't be able to run.

  If all system components are initialized properly, we initialize few arguments like the frame rate controller and the previous/current/next game state and the game loop is started. Note that the previous code should never be reached again.

  Upon exiting the game loop, all the devices that were allocated should be released before exiting the application. For example, if a video device was created, it should be released in order to free all its allocated resources on the video card. Also, if an input device was allocated during the initialization stage, it should be released upon exiting the application. This would be the final code part of the game or the application, and care must be taken to make sure every allocated resource is released.

- **Game loop**
  After initializing all the hardware related components, the application will enter the game loop, and will obviously keep looping until the user decides to quit the application (or for any other game logic reason). This loop is where all the actual game logic resides. It consists of:

  - Loading the current state: Loading all the state's data (unless it is being restarted.)
  - Resetting the frame rate controller
  - Initializing the current state (Make all the state's data ready to be used for the first time)
  - State loop, which keeps looping until the game switches to a different state or if the current state is reset.
  - Freeing the current state upon exiting the state loop
  - Unloading the state data in case it isn't being restarted

  Please refer them to the game state manager for a detailed description of states' functionalities.

- **Handling game states**

  - State switching

    When the game switches to a new state, it remains in this state until it needs to switch to another one. There are many reasons why a game should switch to another state, like the completion of a level, losing, or simply by pressing a cheat key. Restarting the same state is also considered a state switch, but special care is taken not to unload/reload the same data when a state is restarted (Please refer to the "Restarting" section).

    In order to do that, a state should have a loop on its own which is embedded inside the game loop. This loop is called the state loop. The condition to break out of the state loop is to set the next wanted state indicator to a state different than the current one. By doing so, the state loop will break the next time it restarts and checks the loop's condition. Note that when the next wanted state indicator is changed, the game continues running the current state until the end of the state loop (or in other words until the end of the current frame). The state switch will only occur at the next state loop condition check.

  - Loading/Unloading state data

    Each state uses different backgrounds, sprites, animations and sound effects... Therefore each state has its own set of data which should be loaded upon entering and exiting a state (and not restarting it). Data loading/unloading is usually slow, because it needs to load disk files which will leads to excessive access to the hard disk. Therefore it shouldn't be done during gameplay or during "interactive time" where having a constant and relatively high frame rate is extremely important, which can be in a loading or introduction level.

    By looking at the general game flow, notice that loading/unloading states' data is done outside the state loop. During the state loop, all the CPU and GPU power should be devoted to update and render the game, and none of it should be wasted on loading/unloading data like textures and sound files etc..

  - State restarting

    Restarting a state is treated the same way as switching to a different state during the first steps. The next state indicator can be set to "RESTART" for example, which is obviously different than the name of the current state. This will cause the state loop to break. Unlike switching to a different state, some condition checks will prevent the game from unloading the state's data upon exiting the state loop, and eventually they will also prevent reloading the same state data at the beginning of the game loop. These

simple checks will save lots of time when restarting the same state, because loading/unloading can take a considerably great amount of time, especially if the state contains lots of animations and sound effects...

The only thing we have to do when restarting a state is calling its free and initialize functions, which make the state's data ready again for use for the first time. Note that resetting the state's data is not specific to restarting a state, so it doesn't need any special condition checks. (Refer to the game state manager for more info about states' functionalities)