CS 529

Fundamentals of Game Development

1. RTIS?

1.1. Introduction

- A video game is a real-time interactive concurrent events application simulating a starship fighting enemies in space

1.2. Why *concurrent*?

- The starship, bullets, enemies, sound effect, music, and text exist and move at the same time as a series of coincident events
- The list of coincident events:
  - Testing the keyboard for up, down, left, and right key strokes
  - Scrolling the background
  - New ship position calculation
  - Testing if the fire key is hit
  - Creating a bullet
  - Calculating the new bullet position
  - Testing for collision between each bullet and each enemy
  - Calculating the new enemy position
  - Playing a sound effect when an enemy collides with a bullet
  - Playing the explosion animation when an enemy intersects with a bullet
  - Updating the texts
  - Playing the music
  - Etc..

1.3. Why *interactive*?

- The player decides when and where to move the ship
- The player decides when and where to fire a bullet
- The game AI decides when to attack
- The number of bullets increases each time a bullet is fired
- The score updates each time an enemy is hit
- The enemy explodes when hit by a bullet
- Etc...

1.4. Why *real-time*?

- When the ship moves, its new position is calculated at run time
- The collision between the bullet and the enemy is detected at run time
- The enemy's new position is changed at run time

- The text is updated at run time
- Etc...

1.5. How do we write an application with concurrent events?

- We need to be able to execute several instructions at the same time
- It would be nice if we have a CPU dedicated for each event
- Usually, we only have one CPU (or a dual/quad core CPU)
- But we need to execute several instructions in parallel
- Let's divide the second into 60 pieces
- Each piece would be 1/60 or 0.016 a second (16.66 milliseconds)
- If during each 16.66 ms we update sequentially all game components, the game components would be updated 60 times a second
- The player will get the illusion that the events are coincident or parallel – which means that the events are happening at the same time
- In reality the events are not parallel; they are pseudo-parallel
- The events do not happen at the same time, they happen sequentially
- Because the events are updated at a fixed interval of 60 times a second, the illusion of concurrent events is achieved
- Each 16.66 ms duration is one game iteration
- Since the iteration repeats as long as the game is running, the concurrent events are controlled by repeating the game iteration through a game loop
- The game loop also makes the series of events update as a motion picture or pictures in motion

1.6. Game loop

- The game loop iteration duration greatly affects the illusion of concurrent events
- If the duration of the game iteration is long, let's say 0.1 s, then the simulation will feel slow
- Why? Because the reaction to the events happens only 10 times in one second
- On the other hand if the duration of the game iteration is short, like 0.016 s, then the simulation feels smooth
- Why? Because the reaction to the events happens 60 times in one second
- Consequently, the duration of the game iteration is called the *frame time*
- Therefore, the game speed is measured by frames per second
- For example, when we say a game speed is 60 frames per second or 60 f/s or 60 fps, then the game iteration duration is 16.66 ms

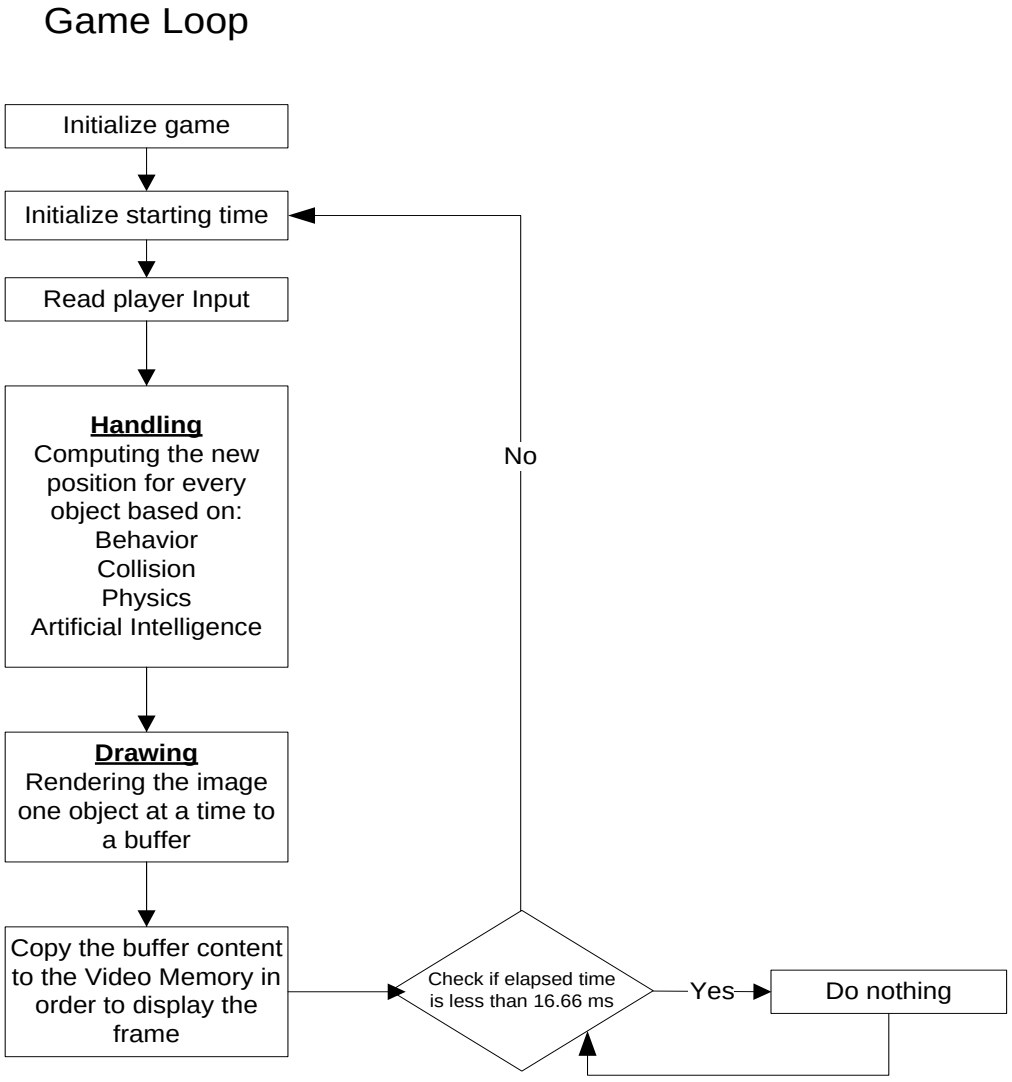
1.7. How do we add interaction at real time with concurrent events?

- During the game iteration, we:
  - Detect and register the user input
  - Execute the behavior of each object; usually the object behavior depends on:
    - Input from the keyboard
    - Input from other objects
    - Physics
    - Collision status
    - AI

    - Etc...
  - Once all the objects are updated, their position and status at the current game loop is determined
  - Render the objects
- This means that the objects are rendered as many times a second as the game speed
- For example in a 60 fps game, the objects are rendered 60 times a second
- During the beginning of the rendering during the game loop, a blank frame is prepared
- Then all the objects are rendered sequentially
- When an object moves, its position at game loop *n* is slightly different than its previous position at game loop *n*-1; also, its position at game loop *n*+1 would be different than the position at game loop *n*

1.8. Game flow overview

- Overview of the game flow:



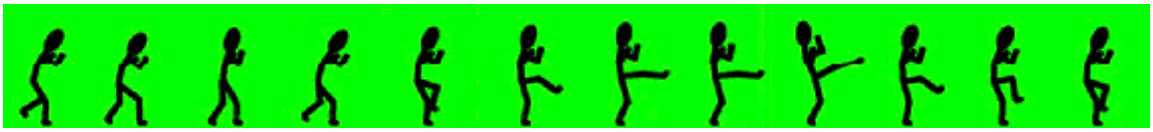
1.9. Motion Pictures

We all go to the movies, but did you know that they were once called "motion pictures"? When you go to a movie, what you actually see is a linear sequence of static images. What does that mean? "Linear sequence" means in a straight line, and "static images" refers to images that are not moving. A motion picture, therefore, is actually a set of non-moving pictures; however, your brain is tricked into thinking that they are moving. This is because the images are flashed so fast that your eyes interpret them as movement, or motion. Additionally, each image is slightly different than the previous one. In other words, image number *n* is slightly different than image number *n*-1.

Motion pictures move at a constant speed of 24 frames per second. This means that during the course of one second, 24 static pictures flash, in sequence, in front of your eyes so your brain interprets it as movement. That means it takes almost 1.5 feet of film for each second of a movie

that you watch. If you ever get a chance, take note of how large the reels of film in a movie theater are.

The stick figure below, for example, demonstrates how a kick would look as a few static frames:



The above image consists of 12 frames. This animation would be about half a second of a movie since a movie displays 24 frames per second. Even so, your brain does not see the individual pictures; instead they move so fast that your eyes "paint" a moving image on your brain that appears as movement.
The same rules apply in video games. We use single frames to act as static images of a character or object in a game and connect a sequence of them together to create motion. We refer to the motion in games as animation.

1.10. Motion Pictures vs. Games

Games are also made of sequentially displayed pictures. However, the images used to create the illusion of motion in a game cannot be determined in advance. Why not? The reason is that each player plays the game differently. For example, one player might start walking toward the left in a specific level, while another might start walking toward the right. Consequently, each time somebody plays the game, different scenarios will happen. Using this fact, we can conclude that the pictures used to create the motion effect in a game have to be constructed while playing the game in order to reflect the user's interaction and perspective.

Movies display 24 pictures per second, while games usually display 60 pictures per second. This means that we have to construct and display 60 pictures per second. Creating the images, flashing them on the screen as needed, and presenting a separate experience to each player is called creating a **real-time interactive simulation**.

- Each time a game loop is executed, one frame (image, picture) is generated and drawn on the screen
- Executing 60 game loops per second will consequently generate the needed 60 frames per second
- Displaying each generated frame for a small amount of time (1/60 seconds) will create the illusion of motion
- Example:

    - To show a ball moving over a background, we first calculate the new position of the ball and then display the ball at its new position. The calculation and displaying are made in one game loop. By repeating the game loop 60 times per second, the ball appears to be moving.

      At game loop *n*, we have a ball on a background positioned at *P(300,300)*. The handling of the ball consists of calculating the new position based on a vector

V(3,0). Vector *V* means that the ball's position is incremented by 3 along the *x*-axis and incremented by 0 along the *y*-axis.

- At game loop $n + 1$, the ball's position is $P(103,300)$.
- At game loop $n + 2$, the ball's position is $P(106,300)$.
- At game loop $n + 3$, the ball's position is $P(109,300)$.
- At game loop $n + 4$, the ball's position is $P(112,300)$.
- At game loop $n + 5$, the ball's position is $P(115,300)$.
- And so on.

- At the beginning of each game loop, the display screen is deleted, then the background is displayed, and finally the ball is displayed according to its new position on top of the background. In the example below, notice that the ball's position changes ever so slightly from frame to frame. These steps are repeated at every game loop, 60 times per second.

