Game Programming Exam Review

Quiz 1 worksheet:

1) .cpp / .hpp -> Preprocessor (add include dirs) ->
   expanded source -> C++ compiler -> object code ->
   Linker (add Lib dirs / dependencies) -> program .exe

   Source code >> Preprocessor (add include dirs) >> expanded source code >> C++ compiler >> Object code >> Linker (add Lib dirs / dependencies) >> .exe

2)
   a) Linker step - config property: Linker >> Input >> additional dependencies
   b) Preprocessor step - config property: C/C++ >> General >> Additional Include Directories
   c) Linker step - config property: Linker >> General >> Additional Library Directories
   d) Compiler error ??

3) PATH=../../SFML-2.6.0/bin;$(PATH)

4)
   a) Need a declaration / prototype for setup()
   b) Either add “using namespace sf” at the top or type this “sf::VideoMode” and “sf::RenderWindow”

5) enum class suits {Spade, Heart, Diamond, Club};

6) double myArr[10]; (takes up 80 bytes)

7) While window.open >> handleInput >> update >> drawScene

8)
   a) Update section of the loop
   b) ??

9)
   a) Avoids seeing a frame being drawn in progress. Current frame is visible while the new frame is being ‘drawn’. Then the new frame is swapped and displayed as the current frame
   b) Front buffer - current visible frame
      Back buffer - frame that is being ‘drawn’ - not visible

10) Event Queue pattern - detects key inputs from the player

Quiz 2 worksheet:

1)
   a) Int* x = new int;
   b) int* y = &b;
   c) int* z = &x;
   d) x = 500;

2) Stack contains
   a = 100
   b = 200

   Heap contains
   x = address to b = 500
   y = address to b = 500
   z = address to b = 500

3)
   a) Memory leak
   b) delete x;
      x = nullptr;
      (do the same for y and z)

4)
   a) std::cout << val;
   b) std::cin >> val;

5) Both can be printed in the console or to a file
   std::cerr is not buffered - used for error info
   std::cout used for regular data

6)
   a) #include <iostream>
      #include <fstream>
   b) Stream is open

7) class by default members are private
   struct by default members are public

8)
   a) override
   b) virtual FloatRect getBounds() = 0;
      Called a virtual function

c) class Goblin : public GameObject
{
private:
	// code
public:
	FloatRect getBounds() override;
}

9)

a) Decoupling pattern / Object oriented programming
b) Main.cpp
int main()
{
  // Declare an instance of Engine
  Engine engine;

  // Start the engine
  engine.run();

  // Quit in the usual way
  return 0;
}

Engine.h
class Engine
{
private:
	// code
public:
	Engine();
	void run();
}

Engine.cpp

// code

void Engine::run()
{
  // Timing
  Clock clock;

  while (m_Window.isOpen())
  {

```
            Time dt = clock.restart();

            // Update the total game time
            m_GameTimeTotal += dt;

            // Make a decimal fraction from the delta time
            float dtAsSeconds = dt.asSeconds();

            // Call each part of the game loop in turn
            input();
            update(dtAsSeconds);
            draw();
        }
    }
```

10)

a) Singleton pattern
b) Allows for only one instance of textureHolder
c) Static instance
d) Game saves
e) It's globally available - makes it harder to debug

11)

a) Update pattern keeps track of the games state and makes sure everything that needs to occur per frame happens correctly
b) Observer pattern:

subject
    Observer collection

observer
    notify

concreteObserverA
concreteObserverB

Quiz 3 worksheet

1)

a) public = can be accessed outside the class
b) private = can only be accessed by the class's functions
c) protected = cannot be accessed outside the class except by inherited classes

2)

a) ~ZombieState()
b) Destroy the object created by the constructor when it goes out of scope
c) Virtual

3)

a) When a smart pointer object is deleted the heap memory is freed.
b) std::unique_ptr is a single reference. Cannot be copied, but can be moved

4)

a) handle - arguments: Input input
   Moves the character based on button inputs
b) update - updates all of the game objects in a scene
   Ex. location, transformation, collision, spawn
c) enter - arguments object (ex character)
   Allows a state to control it's graphics for example
d) exit - method called before switching to a new state

5)

a) States = attack, hit, flee, death
b) Attack -> hit (triggered by player hitting zombie)
   Attack -> flee (triggered by health dropping below 25%)
   Hit -> flee (triggered by health dropping below 25%)
   Hit -> death (triggered by health dropping to 0)
   Flee -> attack (triggered by time going above 2 seconds)
   Flee -> hit (triggered by player hitting zombie)
c) Enter / exit would be useful for the attack and flee subclasses
   Because the animations and what the zombie was doing would change

6)
```
class zombieCommand()
{
public:
        virtual void execute() = 0;
}

class zombieFlee() : public zombieCommand
{
public:
        virtual void execute()
        {
                std::cout << “Run away!" << std::endl;
        }
}
```

```
class zombieAttack() : public zombieCommand
{
public:
        virtual void execute()
        {
                std::cout << "ATTACK!!!" << std::endl;
        }
}
```

7)

a) For AI controlled, pass in an aiActor object to the execute command. This makes it possible to define various types of aiActor objects and pass them on to the same command structure.
b) The commands are stored in a list in reference to the 'current' command.As commands are executed, they will be appended to the list. It's important to keep track of where on the command list the pointer is located. This will affect undo / redo / current command locations.

8)

a) The flyweight design pattern would help reduce the memory footprint by using the same parts (such as graphic) for all of the bees and separating out the different parts (such as size, location).
b) class BeeGraphic
        private:
                Texture beeTexture
                Sound beeSound

    class BeeUnit
        private:
                BeeGraphic* beeGraphic
                Vector2f beelocation
                int beeSize
                int beeHealth