Test 1, CPSC 4160/6160 – Spring 2021, due on 3/1, 11:59pm. Submit through Canvas, questions posted to Piazza, public or private. **Do not disclose any answers on Piazza**.

Answer the following in the space that is given. You should not need extra space if you are careful about your organization and planning**. I recommend you write in a draft document and only fill in the pdf when you are complete.** Each question is 8pts each, totaling 40pts.

1. **A)** From what you have seen thus far in class, what are the pros and cons for using SDL to build a 2D game engine upon? Describe at least two positive and two negative aspects, along with any neutral remarks that stand out. **B)** Describe in a sentence or two the game mechanics of a specific game you might like to implement with your game engine if you have time. How well does your game map the mechanics to the game components we have described thus far (game loop and mechanics, user input, game objects, player sprites and control)? Describe each along with any need to adjust the generic components to meet the needs of your game.
2. **SDL can be an extremely powerful tool for 2D game creation for a multitude of reasons, with some drawbacks that come with using any tool. The biggest pros for using SDL in my eyes have been its extreme versatility among which operating systems or devices/hardware can utilize it. SDL supports Windows, Linux, Apple devices (Mac and iOS) as well as Android and can be implemented on any of the above rather easily. Another pro, and arguably my favorite part of SDL, is that it is open source. All functionality, vocabulary, syntax, and much more is available online to research and learn with no restrictions. Working on a project for school or as a hobby and being able to access any information you need within a few clicks is extremely helpful.**

**As for the cons of SDL, I have found it frustrating to set up on Windows accompanied with its lack of designated IDE for development. If SDL had an IDE designed for it that made development more streamlined, it would become even more powerful and user-friendly. The other con I have found is that SDL can be very low-level and difficult for a beginner to pick up. This is the first tool I have used for 2D game creation and after class sessions and doing my own research I have discovered there are much easier-to-pick-up tools out there for someone just getting into game development like this.**

1. **My favorite 2D game of all time is Terraria. I have poured hundreds of hours into the game learning every detail and I think the part I most enjoy and would love to implement into my own game engine would be the class/armor system. Each “class” has its own distinct characteristics depending what armor you wear and nothing else. No class is permanent and can be swapped out by simply switching to a different set of armor with different attributes. User inputs get switched when using different classes, sprites gain new effects, and player control changes in both speed and fluidity depending which class’s armor you have equipped. The game’s classes get diverse to the point that enemies will even behave differently against different classes and will change their style depending on what you are. My game would need an extremely in-depth class system with specific functionality for weapons and enemies alike to implement a similar system.**
2. In the lectures, we have limited our discussions to fixed time frame rates (60fps). Describe in detail the differences necessary to build a game with *variable* framerate, that is a framerate that is allowed fluctuate. What are the advantages and disadvantages of a variable framerate?

**Implementing a variable frame rate in a 2D game as opposed to a fixed frame rate requires the game engine to continuously calculate the time since it last calculated a tick and update its character according to that and the system’s performance. A fixed frame rate will also calculate the time since the last tick but will then either tick at a specified time or wait until the next update to tick. Fixed frame rates most often require the game loop to tick specific functions a certain number of times and will keep the gameplay consistent and smooth at its fixed fps. A variable frame rate may end up ticking the same function more times or less depending on the state of the game and the character’s interaction. Where a fixed frame rate engine may call the “UpdateEnvironment()” function three times per loop, a variable frame rate’s engine may have interactions causing it to update five times or even less to one time.**

**The advantage of using a variable frame rate is that it allows the system running the game to run at its maximum performance levels. A game with variable frame rate on a powerful system may be able to run a game at 144 fps and have much smoother and cleaner gameplay than if the engine were locked to 30 or 60 fps. On the same note, the disadvantage of a variable frame rate can be the exact opposite. A poorly optimized game or a low-end system on a variable frame rate engine may see more stutters and tearing during gameplay because the system is struggling to hold its frame rate and a consistent number while it strives to go higher. Alongside this, using a variable frame rate with a struggling system can cause other aspects of the game to freeze up or stop implementing inside the game as intended. An object that should be moving smoothly or updating regularly may not look or perform as well as if it were hooked to a continuous, fixed rate system.**

**Many game companies and indie developers have been utilizing ways of allowing the user to choose whether they wish to have variable frame rates or fixed frame rates. Users with powerful systems will most often choose to completely unlock their frame or to fix it at the highest possible value. Other users will fix their frame rates to the refresh rate of their monitor, or even simpler, enable a setting called vsync which automatically fixes your game’s frame rate to that of your monitor to limit tearing and obstructions in the gameplay. Lower end systems make use of this very efficiently.**

1. **A)** Perform a mini research and discovery effort (no more than one hour or so) and define here (in your own words, in no more than one paragraph) what is the object-oriented programming design pattern called a "*Singleton*"? **B)** Next, is the GameEngine object class we describe in this course a Singleton? Why or why not? **C)** In your own opinion, is the Singleton class a good choice for building the core class of a 2D game engine? Defend your opinion with supportive evidence in your own words.
2. **A “Singleton” design pattern is a pattern that allows for a program to create one and only one instance of a class which is the sole controller of its instance. Singleton’s make it so that a class can only have one instance operating at a time, and the operational class is also the only one responsible for itself. For further detail, an instance of a Character class will have a hidden, private constructor so that it can never be created from outside the class, and the Character instance can be easily accessed with a global singleton “get” function. The overall advantage of having a singleton object is so that there is only one of them active in the program, it is the only thing that can control it, and it can be accessed globally in the program.**
3. **I would say that yes, the GameEngine object class from this course is a Singleton because we only want one instance of the engine in our game, we don’t need outside sources creating another game engine, and we want it to be accessible by everything in our game to increase functionality and cooperation. Our goal has not been to implement multiple GameEngine objects in each program, but to have one instance that controls the flow of the game.**
4. **I think the ultimate choice for implementing the core class of a 2D game engine as a Singleton or not is entirely dependent on the mechanics of the game itself, but I definitely believe it would be a good decision to do so. One of, if not the most, important parts of any game is the core class containing the character or object with which you play through or control. If you could limit your game to having only one instance of that character which can be accessed globally by anything else implemented, while also having the security that nothing else will be able to create it or destroy it, I believe it is a solid option. I can only imagine the frustrations coming about because the core class was not a Singleton and was therefore not accessible by something else in the game, and the workaround necessary to implement that mechanic. If the core class had been a Singleton it would have been easily accessed, changed, or upgraded depending on the mechanic with little interference due to class constraints.**
5. We learned about the character input for games in SDL primarily through keyboard input. Imagine your first job in game programming was at a mobile game studio that wanted to use the touchscreen for a tablet or phone as input for a game. What are the primary differences you must address in gathering input from a touchscreen vs keyboard input? What adjustments are necessary from the programming side? What limits or benefits will arise on the game design side? Is SDL a sufficient library for the changes, why or why not?

**Gathering input from a keyboard is a relatively simple matter once you get the functions down and learn to handle the input. Keyboard inputs have two states: Key Down and Key Up. Alongside this, any keyboard input can be directly traced to a specific key using SDLK\_{key} which helps streamline input and make it easy to call functions dependent on specific keys. Touch input can be a lot more complex. A touch screen implementation in SDL can require multiple textures to be overlayed over certain areas that will receive the touch input and call functions accordingly. There is no “master key” to call a function. To add on to that, touch inputs will require the programmer to take note of a lot more things: touch location, texture being touched, touching down on the screen, touching up on the screen, touch motion, and even multiple touch inputs at once. Touching down and touching up are the same actions as Keying down or keying up, but the twist comes with touch motions, specific textures, and multiple inputs. Touch motions will require the programmer to implement an input handler like that of a mouse’s input. The input will start in one location and end in another, calling different functions or triggering events at press, along the motion, and at release. As for handling multiple inputs, some sort of I.D. system will need to come into play to identify and handle each input at one time.**

**SDL can be a sufficient library for handling touch screen inputs. The SDL library has functions for handling touch input such as “SDL\_FINGERDOWN”, “SDL\_FINGERMOTION”, and “SDL\_FINGERUP” which will handle the necessary events of a touch screen. The difficulty will be locating the touch on the screen and implementing a method for handling actions in that area. Utilizing touch inputs for a 2D game can be massively beneficial for the mechanics of the game. Touch screens unlock a dimension of inputs previously not available to games. With touch inputs, users will be able to manipulate objects and interact with the world or characters in ways we could not think of 10-15 years ago. On the flip side, using a touch screen means losing the precision and benefit of knowing that every input we do will result in a specific action. It can be easy to touch the wrong part of the screen, slide your finger too quickly, or mess up an input, and it can also be hard to program for each input type and location.**

1. Imagine a side scroller game that has a werewolf hero. Our hero will be able walk, kick, and jump in its human form while it has additional special bite, howl and climb abilities when it is a wolf. Our hero is only allowed to kick when human. Design a player control for this character. Include your choice of the control metaphor (or multiple) and a state machine (or multiple) for the behavior along with descriptions of the transition triggers\* that move between states. Support why you opted for this style of player control for this character. (\*that which *triggers* the transition)

**An optimal method for controlling this hero would be through the use of two different player classes, a state changer that switches the hero’s form once events are triggered, and two different state machines for handling the hero while in human form and in werewolf form.**

**To begin, the hero’s default state is human form, where the player will have velocity movement control but only on the x-y plane. That is, the human form of the hero will run forward/backward while the left or right arrows are pressed and will jump straight up in the air when the up arrow is pressed. Movement will stop immediately in the left/right direction if the input buttons are released. While in human form, the user can press ‘F’ to kick, which stops the hero’s movement in any direction (even upwards and downwards) to perform a kick in the direction the hero is facing. After the animation the hero will be stationary on the ground (or continue up or down) until further input. To jump, the user will press the up arrow which cancels the hero’s momentum on the x-plane and jumps straight up to double the character’s height before coming back down to the ground. The hero’s human form can only press the left or right arrow one time while in the air after a jump, moving the character an increased 2 player lengths to help with jumping towards platforms.**

**While the hero is in werewolf form, its control metaphor becomes velocity-delta movement where the longer the movement key is pressed the faster the hero will move (up to a max) and will slow down to zero upon release or opposite inputs. To add onto this, the hero’s werewolf form will be able to jump diagonally forward or backward using the up arrow with jump length depending on the speed of movement. The werewolf form loses the ability to kick with ‘F’, but instead gains the abilities to bite, howl, and climb. The bite is a forward-lunging attack that is now activated with ‘F’. Biting will shoot the hero forward 3 player lengths and kill any enemies in its path with a cooldown of 5 seconds. Pressing ‘D’ in werewolf form will activate its Howl ability which ‘terrifies’ all enemies and stops all movement and projectile movement on the screen for 3 seconds with a 10 second cooldown. For even more movement, the werewolf can climb up wall surfaces by holding the direction arrow of the wall and the ‘Space’ key at the same time. Letting go of ‘Space’ while climbing will see the werewolf begins sliding towards the ground slowly.**

**The trigger for transitioning between states and switching state machines will be on a timer in the game loop. The game will have day/night cycles that last for a specified number of frames, with the sun and moon traversing through the sky over that number of frames to signify when day or night is active. During the day, while the game engine loop is showing the sun in the sky, the hero will start or transition into human form immediately. As soon as the sun disappears and the game engine transitions to showing the moon in the sky, the hero will immediately transition into a werewolf and the state machine will change all inputs to the werewolf’s control methods.**

**This style of player control fits a two-class system with different abilities and fits the theme of controlling a hero that can transform into a werewolf. Werewolves are inherently stronger and only come out when the moon is out, so having a hero transition to werewolf at night and gain better movement with stronger abilities only seems fitting for this type of game. It also removes the disadvantage of a user having to press a button to change states and instead helps focus on controls.**