**Weight: 40%**

**Table of Contents**

[Objective 1](#_Toc398385818)

[Controls 2](#_Toc398385819)

[Gameplay 2](#_Toc398385820)

[Menus 2](#_Toc398385821)

[Game Window 2](#_Toc398385822)

[Architecture 3](#_Toc398385823)

[Game State Machine 3](#_Toc398385824)

[Game Entities 3](#_Toc398385825)

[Events & Messages 3](#_Toc398385826)

[Graphics & Sounds 3](#_Toc398385827)

[Design 4](#_Toc398385828)

[Gameplay State 4](#_Toc398385829)

[Collision Detection & Response 4](#_Toc398385830)

[Winning 4](#_Toc398385831)

[Losing 4](#_Toc398385832)

[Game Over 4](#_Toc398385833)

[Menus 5](#_Toc398385834)

[Main Menu State 5](#_Toc398385835)

[Options State 5](#_Toc398385836)

[How to Play State 5](#_Toc398385837)

[Credits State 5](#_Toc398385838)

[Pause 5](#_Toc398385839)

[Features 6](#_Toc398385840)

[Repeatable 6](#_Toc398385841)

[Basic 7](#_Toc398385842)

[Intermediate 9](#_Toc398385843)

[Advanced 11](#_Toc398385844)

[Deductions 12](#_Toc398385845)

[Submissions 12](#_Toc398385846)

# Objective

Create a game in a genre of your choice with player, computer, and projectile entities.

# Controls

## Gameplay

* The keyboard (and/or mouse) controls the player entity's movement.
* The Escape key pauses / unpauses the game.
* All keys must be listed in the README (see Submissions for details).

Menus

* The Up arrow moves the menu cursor up one item (without going off the list).
* The Down arrow moves the menu cursor down one item (without going off the list).
* The Left arrow decreases the value of the menu item (for volumes and other slider controls).
* The Right arrow increases the value of the menu item (for volumes and other slider controls).
* The Enter key selects the menu item (opens a new menu or starts the game).
* The Escape key moves the cursor to the Exit item or returns to the main menu.
* Any additional menu keys must be listed in the README (see Submissions for details).

## Game Window

* Pressing Alt + Enter will toggle the game between fullscreen and windowed modes. This code should be in Game::Update (not the individual states). (See for more details.)

# Architecture

## Game State Machine

* The Game class handles switching between states.
* The states are derived from a base class.
* A Main Menu state handles the main menu.
* An Options state handles the options menu.
* A How to Play state handles the how-to-play menu.
* A Credits state handles the credits menu.
* A Gameplay state handles the gameplay.

## Game Entities

* The game entities are derived from a base class.
* All entities are stored in the Entity Manager.
* All entities have a reference count (to eliminate memory leaks by using AddRef and Release).

## Events & Messages

* The game uses the Event Manager and/or the Message Manager.
* At least one event or message is triggered and responded to.
* Events and messages must be useful, not just triggered when a key is pressed.
* Messages should be used if a game entity is creating another game entity or destroying one, in order to interact with the Entity Manager when it is not iterating.
* Events should be used if a game entity is interacting with another game entity without knowing the specifics of the class (does not include the other class's header file).

## Graphics & Sounds

* The game *cannot* use any asset from lecture or lab.
* All assets have *your* initials on the front of the filename (e.g. *ABC\_player.bmp*).
* A bitmap font is used to display allmenu and gameplay text, using the Cell Algorithm to calculate the character's source rectangle within an image. The game title can be displayed with a fixed image. Debug output does not require a bitmap font.
* All sound effects are in *.wav* format. All background music files are in *.xwm* format.
* The game runs at a resolution that is supported by fullscreen mode (use the Graphics Manager's Resize method to switch between modes). The classroom projector runs at 1024x768.

# Design

## Gameplay State

* The Gameplay state initializes and runs the game logic.
* A player-controlled entity moves based on keyboard input (can also support mouse input).
* A computer-controlled entity moves automatically in any pattern:
* Chases the player entity
* Avoids the player entity
* Moves randomly
* Moves in a predetermined pattern
* Projectile entities are fired from the player entity or a computer entity.
* Projectiles are spawned at the firing entity's position.
* Projectiles' velocities are determined by the orientation of the firing entity.
* The player's score is displayed while the game is playing.
* Background music is played during the game.

## Collision Detection & Response

* The player entity cannot move off the screen, though could wrap around the screen without ever being invisible.
* Computer and projectile entities can move off the screen (if desired).
* Collision is detected when the player entity and a computer entity hit.
* Collision is detected when a projectile entity hits another entity.
* A sound effect plays when the player entity takes damage.
* A sound effect plays when the computer entity takes damage.
* The player's score increases when an entity is destroyed by the player's actions.

## Winning

* The player can win the game by accomplishing a specific objective (your choice):
* Defeats all the opponents in a level
* Survives for a specific amount of time
* Reaches the end of a level
* The goal and the player's progress toward the goal is displayed while playing (e.g. 5/10 enemies remaining, 30 seconds remaining, the finish line is 🡺, etc).
* A sound effect plays when the player wins the game and winning text (e.g. "YOU WIN") appears on screen until the user presses any key (or with a timer for at least 3 seconds).

## Losing

* The player can lose the game by one or more ways (your choice):
* Loses all health and/or lives (which is displayed while playing)
* Falls off the level or enters a dangerous environment (pit of snakes, lava, etc.)
* Fails to complete an objective in a specific amount of time
* A sound effect plays when the player loses the game and losing text (e.g. "YOU LOSE") appears on the screen until the user presses any key (or with a timer for at least 3 seconds).

## Game Over

* Regardless of whether the player won or lost:
* The credits state is displayed
* Then the game returns to the main menu state

# Menus

## Main Menu State

* The Main Menu state shows the game's title and menu selections:
* Play - enter the Gameplay state (restarts the game, resetting all gameplay data)
* Options - enter the Options state
* How to Play - enter the How to Play state
* Credits - enter the Credits state
* Exit - close the game
* Master Volume levels are loaded from a file and applied to the game when the state is entered.
* The arrow keys navigate the menu with a visible cursor.
* Pressing Enter will select a menu item.
* Pressing Escape will move the menu cursor to Exit.

## Options State

* The Options state allows master volumes to be increased and decreased:
* Music - range of 0 -> 100 (must display the integer value)
* Sound Effects - range of 0 -> 100 (must display the integer value)
* Exit - enters the main menu state
* Master Volume levels are applied after any change is made.
* A song plays while the cursor is selecting Music. A sound effect plays while selecting SFX or when adjusting the SFX.
* Master Volume levels are saved to a file when the state is exited.
* The arrow keys navigate the menu with a visible cursor.
* Pressing Escape will enter the Main Menu state.

## How to Play State

* The How to Play state shows the objective, controls, and game entities:
* Objective - how to win
* Controls - how to move / shoot / interact
* Player - displays and briefly describes the players and their abilities (if any)
* Enemies - displays and briefly describes enemies and their vulnerabilities (if any)
* Pressing Escape will enter the Main Menu state.

## Credits State

* The Credits state shows your name and *Structure of Game Design at* *Full Sail University.*
* The credits should include anyone who helped with the game or assets.
* Pressing Escape will enter the Main Menu state.

## Pause

* The Pause menu should be embedded within the Gameplay state, not a separate state.
* Pressing Escape in the Gameplay state will pause and unpause the game, but not during a game over (while "YOU WIN" or "YOU LOSE" is displayed).
* Game entities do not move while the game is paused (but can animate).
* The Pause menu shows its selections on top of all the game entities:
* Resume - continues the Gameplay
* Quit - switches to the Main Menu state
* The arrow keys navigate the menu with a visible cursor.
* Pressing Enter will select a menu item.

# Features

## Repeatable

* Animations (4 maximum)
  + Uses the Cell Algorithm to determine each frame's source rectangle or the Anchor Point Algorithm to store each frame's source rectangle and anchor point.
  + Must have at least 3 frames.
  + Used on any entity or separate entities.
  + Each animation must be a separate image (one image for walking, one for attacking, etc.).
* Additional Levels (+3 maximum)
  + Must have a distinct appearance (different background or level layout).
  + Must introduce new gameplay. Could be a new character or weapon to use. Could be a new type of enemy or a different pattern of enemies. Not just more entities.
  + Must display the level number or name (e.g. Stage 1, Water World) for 3 seconds at the start, while the game is loading (should wait for 3 seconds or until the user presses a key).
  + Must display the level number or name on the Pause menu.
  + Must have separate functions (or config files) to initialize the levels.
* Additional Enemy Types (+3 maximum)
  + Must have a distinct appearance (different color or image).
  + Must have different behavior (e.g. one moves randomly, another chases the player, etc.).
  + Could have different attributes (e.g. one is slower/stronger and the other faster/weaker, etc.).
  + Each must be displayed and briefly described in the How to Play state.
  + Must have separate classes for each enemy type.
* Additional Weapons (+3 maximum)
  + Must have a distinct appearance (different color or image) for the weapon or projectile.
  + Must have different behavior, not just different stats (e.g. a revolver fires one projectile, a shotgun fires a spread of projectiles, a sword is a melee cone, etc.).
  + If the user can switch between weapons, the active weapon must be displayed.
  + Each must be displayed and briefly described in the How to Play state.
  + Must have separate classes for each projectile type or separate functions for firing each weapon.
* Additional Characters (+3 maximum)
  + Must have a distinct appearance (different color or image).
  + Must have different attributes (e.g. one is slower/stronger and the other faster/weaker, etc.).
  + Could have different abilities or attacks.
  + Each character must be able to win and lose the game.
  + Character Select screen must be displayed at the start of each new game.
  + Each must be displayed and briefly described in the Character Select screen and in the How to Play state.
  + Must have separate classes for each character or separate functions to initialize them.

## Basic

* Intro Screen
  + Must have some kind of time-based effect (e.g. fade in/out, “press enter” blinking, etc.).
  + Must be able to be skipped if ANY key is hit.
  + Only occurs when the game FIRST starts up, not each time the main menu is displayed.
* Tally Screen
  + End of level summary that counts up score.
  + Displays player performance (e.g. enemies killed, hit percentage, time, etc.).
  + Must use a timer to increase the score over time (i.e. watch the tally count up 10 points each fraction of a second to reach the final score).
* Scrolling Credits
  + The credits scroll from one edge of the screen to the opposite (bottom to top or right to left).
  + The credits wrap around to repeat after they all scroll off the screen, or they stop when the first credit reaches the edge of the screen so all are visible, or the state automatically exits to the Main Menu when the credits are off the screen.
  + The scrolling resets each time the Credits state is entered.
* Slide Show
  + Shows one picture after another (3 minimum) with a time delay to tell a story / describe something.
  + The slide show starts from the beginning each time it runs.
  + Must be able to move to the next slide by pressing ENTER.
  + Must be able to be skipped by pressing ESC.
  + Could be used for a How to Play state with multiple pages.
* Attract Mode
  + After waiting for 10 seconds on the Main Menu state with no input, the game goes into a mode where it shows the How to Play state, then the Credits state, and then back to the Main Menu state.
  + Returns immediately to the Main Menu if ANY key is hit.
* Text Effects
  + The appearance of characters change one-by-one over time (must use the elapsed time).
  + Characters must be changed individually (not the whole string all at once).
  + Text could appear one character at a time.
    - Characters could cycle through various colors, or fade in and out.
    - Characters could shrink and grow, or spin or dance (rotating left and right).
* Progress Bar
  + A bar that fills/depletes to indicate the quantity of something (could be used to show life, mana, power, sound volumes, etc.).
* World Coordinates
  + The game world is larger than the screen and uses World Coordinates for all entity positions, collision data, etc. in the game.
  + The game world has a 'camera' position which is used as the relative origin for all entities to render (i.e. the entity subtracts the camera's position from its own to get its screen position).
* Parallax Scrolling
  + Multiple layers (3 minimum) scrolling at different speeds in the opposite direction of the player's movement (i.e. when the player walks left, the layers move right; and vice versa).
  + The layers do not scroll when the player is standing still.
* AI Waypoints (3 minimum)
  + An enemy entity follows a set path from one waypoint position to the next.
  + Waypoints must be created to navigate the map, not just a random location to wander toward.
  + Waypoints are stored when the enemy entity is first created.
* Auto Fire
  + User must hold down a fire key for projectile entities to shoot each fraction of a second (like a machine gun).
* Weapon Cooldown / Recharge
  + A timer prevents projectiles from being fired rapidly, requiring a second or more to recharge.
  + Must be apparent that the weapon is cooling (e.g. a progress bar fills up, an animation plays, or a sound effect plays while cooling).
  + User must RELEASE the fire key to start the cooldown (cannot hold down the fire key).
* Charged Shot
  + User must hold down a fire key for a certain amount of time to charge a shot.
  + Must be apparent that the weapon is charging (e.g. a progress bar fills up, an animation plays, or a sound effect plays while charging).
  + Increases the damage, speed, and/or number of projectiles.
  + Fires the projectile when the user RELEASES the key.
* Mouse Aiming
  + A projectile entity is rotated to move toward the mouse cursor (either when the entity is created, or each frame the entity updates), using vector math.
  + The mouse cursor must be visible.
* Mouse Object Selection
  + Able to click on a game entity to select it.
  + Selected entities must be visibly distinguishable (different color, background, etc).
  + The mouse cursor must be visible.
* Rectangle Object Selection
  + Selecting multiple game entities by dragging a rectangle over the top of the entities.
  + Selected entities must be visibly distinguishable (different color, background, etc.)
  + Must draw the rectangle as it is being dragged.
  + The mouse cursor must be visible.

## Intermediate

* Stack-Based Game State Machine
  + Push one state over another and see them both (i.e. push the Options state onto the Gameplay state and see the options menu on top of the gameplay).
  + Should be able to push an UNLIMITED number of states over each other.
  + Only the top level state receives input; all states on the stack must render (and can update)
* Particle Effect
  + Particles have an image, position, and velocity.
  + Particles fade out after a short amount of time.
  + Must have at least 20 particles in the effect.
* Z-Sorting
  + Within the Entity Manager's RenderAll method, game entities are first added to a priority queue based on a depth value, so they render from back to front.
  + After all game entities are in the priority queue, the queue is rendered.
  + The depth value must be a separate data member.
  + This is useful for games such as Double Dragon in which the draw order changes depending on how far "up" the street the characters are standing.
* Mini-Map / Radar
  + Shows the position of the player and ALL the enemies (though could use fog of war).
  + Alternatively could show the direction of ALL the enemies relative to the player's position.
* Tile Mapping for the Level
  + Tiles are numerical values to indicate which section of an image to render, using the Cell Algorithm to calculate the tile's source rectangle.
  + Uses an array of tile IDs for the background, platforms, dangerous areas, etc. which covers the ENTIRE level (not just individual platforms). Tiles cannot overlap.
* Platforms / Obstacles (3 minimum)
  + Platforms, walls, and other obstacles stop the player's movement.
  + Platforms can be separate entities or held in a level manager.
  + Player must collide with at least three sides (could allow them to jump through the bottom).
* End of Level Boss
  + Spawns when the player reaches the end of the level / defeats all the other enemies.
  + Must attack the player (cannot just sit idle).
  + Must have two distinct abilities to be activated when the boss' health drops below 50% and 25%. (E.g. Boss constantly fires lasers at player. At 50% health, the boss releases minions. At 25% health, the boss creates a shield to absorb damage.)
  + Must have significantly more health than a normal enemy (but not so much health that it cannot be defeated within a minute or two).
  + Health must be displayed on screen (this would be a good use of the Progress Bar).
  + Must be displayed and described in the How to Play state (including what abilities it has and how to defeat it).
* Enemy AI States (3 minimum)
  + Must have different behavior (e.g. chasing, avoiding, idle).
  + Each behavior must have a separate method to run logic (e.g. UpdateChase, UpdateAvoid, UpdateIdle).
  + Must use a switch, function pointers, or state class pointers to call the behavior method.
  + At least one enemy must transition between three states.
* Player State Machine
  + The player has different states which control its movements, behavior, and responses to input (e.g. jumping, attacking, running). Animations do not count as separate states.
  + Each behavior must have a separate method to run logic.
  + Must use a switch, function pointers, or state class pointers to call the behavior method.
* Two Player Simultaneous
  + Cooperative or versus (at same time with different keyboard keys or on separate controllers).
  + Each player controls a game entity and is an active participant in the gameplay, not just an observer.
* Power Ups (2 minimum)
  + An entity that gives the player some kind of benefit (e.g. ammo, health, new weapon, etc.)
  + Player (and/or player's projectiles) can collide with the power up to gain its benefit.
  + A sound effect plays when the power up is acquired.
  + Entities affected by a power up must have a distinct appearance (different color or image).
  + Any duration-based effect must have a timer displayed while running (e.g. Invincibility: 4 seconds remaining).
  + Must be displayed and briefly described in the How to Play state.
* Upgrade Shop
  + Shop screen appears between levels or is selectable in the pause menu.
  + Player can purchase upgrades with currency. Currency must be displayed during the game.
  + Upgrades (3 minimum) must have different effects (e.g. health, ammo, weapons, etc.).
  + The upgrade's name, price, and effect are printed next to the image of the upgrade.
* Cheat Code / Combo
  + Entering a series of key presses (at least 3 keys in a specific order) to perform an action.
  + Must time out if too much time passes between inputs.
  + A sound effect plays when the keys are pressed successfully.
  + Must be described in the How to Play state.
* Debug Mode
  + A mode that shows useful information for debugging the game.
  + Press a key to toggle this mode on/off (usually F1).
  + Must at least show: FPS, number of entities in the Entity Manager, and object data (e.g. collision box, world space position, velocity vector, hit points, damage, etc.).
  + Object data must travel with the game entity (not just displayed on the side of the screen).
  + ALL game entities must show some kind of debug data.

## Advanced

* High Scores State
* Shows the names and scores of at least ten of the top players.
* The names and scores are loaded from a file when the state is entered.
* If the file does not exist, ten default names and scores (low enough for the player to earn a high score) are added to the high scores list (sorted correctly) and saved to the file.
* If the player won the game and their score is high enough for the high scores list:
* The player enters their name (or initials)
* The player's name and score are added to the high scores list
* The high scores list is saved to the file
* After the game, regardless of whether the player won or lost, the High Scores state is displayed (the player can press any key to skip), then the credits state.
* Key Binding
  + Allow the user to rebind the keys to play.
  + The active key binding must be displayed in the How to Play state.
* Mouse Menus
  + Click on any menu item to select it.
  + The mouse cursor must be visible, though could use a custom cursor image.
  + Must still support the keyboard for menu item selection too (the keyboard input has priority if the mouse isn’t moving).
  + Every game state must use mouse input for all menus (volume controls, pause, and exit).
* Gamepad Compatible Game
  + Can use a gamepad to play the game, in addition to the keyboard (the Input Manager has methods to support most gamepads).
  + Everything you can do with the keyboard, except toggling full screen, must be supported by the gamepad.
  + Must pause the game when the gamepad is disconnected or reconnected, if the gamepad was connected when the Gameplay state was entered.
* Save / Load Game
  + Can save player's progress (e.g. score, level, weapons, lives, etc.) and restore it later. This could be just one save slot that auto saves when the player reaches the next level.
  + Must be able to start a new game through the main menu or choose to continue the old game.
  + Must print out some info about the currently saved game (e.g. Level 2 - Score: 9999).
* XML Config File
  + The attributes of a game entity, level, animation, particle effect, and/or input controls are saved in an XML file.
  + Must have three objects (of any type) and at least three different attributes.

# Deductions

**Code**

* Compiler Error -10 points
* Program Crash -5 points
* Memory Leak -5 points
* Missing File Comment -2 points

**Assets**

* Missing Asset -5 points
* Uses Lecture or Lab Asset -5 points
* Music in .wav File -5 points
* Asset Name Missing Initials -2 points
* Unused Asset -2 points
* Missing README.txt -5 points

# Submissions

**README Text File**

* The README file contains simple details about the game:
* The game title and programmer's name
* A list of controls
* The win and lose conditions
* A list of enemy types and how to defeat them
* A list of EVERY feature implemented, briefly describe each feature:
  + List the types and differences of repeatable features (animations, levels, weapons, etc.)
  + List the types and differences when the feature has a minimum (waypoints, AI states, power ups, etc.)
  + List the controls used to activate the feature (fire timer, charged shot, etc)
  + List the entities that implement the feature (sine wave, mouse follow, AI, etc)

**Game Executable**

* The executable is the Release mode build of the game project.
* Place the executable in the folder next to Resources, and run it.

**Project**

* Delete any Debug, Release, and ipch folders (in the Solution and Project directories).
* Delete any .ncb and .sdf files. Delete this document file (if it is within the directory).

Have an instructor completely grade your check-off sheet **FIRST**, and turn in the check-off sheet.

Zip the solution, project, readme, executable, code, and assets. The zip should be less than 15MB. Rename the zip folder to: ***StudentIDNumber.zip*** (such as 123456.zip). Upload the zip file to FSO on the Game Project assignment, and drag the zip file onto the Turn-In folder at [\\studentvfiler\SGD\](file:///\\studentvfiler\SGD\).