Technical Design Document

Nikkolas Diehl – 16945724
Auckland University of Technology – AUT

Logic and Technical Algorithm Descriptions:

- The slow down mechanic will be an algorithm that stores your current position and arm movement as two separate variables and then another two variables as future movement. When any input is received it will update the future movement variables and then the algorithm will check if the future variables match the current position and arm movement. If they do not, it will force an update and increase the deltaTime multiplier that will determine the speed of the game.
    - When no position movement is detected but an arm movement is detected, the algorithm will also get the current arm rotation angle and update the dynamically animated sprite based on that as well as change the deltaTime multiplier from 0 to 0.5
    - When position movement is detected; regardless of any arm movement, the algorithm will again; get the current arm rotation angle and update the dynamically animated sprite based on that, as well as change the deltaTime multiplier from its current state (either 0 or already at 0.5) to 1
    - This multiplier will be parsed into every single entity within the INGAME scene *except* the player and control how fast each entity is processed. From AI, to bullets, to particles, etc…
- The animation algorithm will be used to control how the AI and the player is animated. In both situations, the sprite used will be a large 2D sprite sheet array of movement. For the animated of rotation, 360 slightly altered sprites will be created to show the players body shifting through all 360 degrees of a circle. The animation algorithm will then get the current angle between the crosshair and player entity (shown as $cos \emptyset = \frac{\bar{a}.\bar{b}}{|\bar{a}|.|\bar{b}|}$) (Mykhailo, 2011) (MvG, 2017) and set the animation sprite sheet location to the correct sprite allowing the sprite to have the full range of circular animation.
    - The animation algorithm will also be controlled by the weapon state machine controlling each AI enemy entity and player entity. If the STATE for an example AI is NO_WEAPON and running, the animation algorithm will set the animated sprite sprite location to the correct path on the sprite sheet.
    - If the animation algorithm detects the state as HOLDING_SWORD then the sprite will change to a sprite holding a sword whilst still retaining the full animation circle.
- The collision detection algorithm will be diverse and action driven. Each bullet, player, and AI will collide with walls, floors, roofs and each other. Every particle will collide with the walls, floors and roofs but will not collide with each other.
    - A thrown weapon when collided will change the AI or player state to STAGGERED
    - A sword attack when collided will kill; as will a bullet.
    - A bullet hitting a wall will spawn a small number of particles as a spark and the bullet will no longer be drawn or processed.
    - A small number of blood particles will spawn on death of AI or player and will remain for the entire duration of the mission, colliding constantly with the level and

players/AI walking through it. AI enemies that are dead will be destroyed (hidden and no longer processed/drawn) on death. Particles will spawn in place of the dead body.

- Any thrown weapons or spawned weapons will collide with everything around it.
- All particles, AI, players, and bullets will obey gravity and fall with it.
- Algorithm for testing a Line, Line from online source (Line-Line intersection, 2019).
- Algorithm for testing the distance between two points from online source (Christopher, Rama, & Baiherula, n.d.)

- A state machine will be used to control the AI and players. There will be a limited amount of states such as STAGGERED, NO_WEAPON, HOLDING_SWORD, HOLDING_GUN, DEAD and ALIVE. The player and each AI will have two current states. One to check if they're alive or dead and one to check their current movement state as one of the other 4 states.
- The AI algorithm will be relatively simple.
  - Each AI will be able to shoot, attack, and move through walls as the walls will be seen as *background*.
  - Each AI will be able to shoot upwards at you and downwards through floors and roofs but will not be able to move downwards through floors or upwards through roofs.
  - Each AI will constantly LERP *only* on the X axis towards you. This allows gravity to constantly control its Y movement and make a physics-based game whilst the AI will constantly try track towards you whether your bellow, above or on the same X plane as you.
  - When an AI is within a few pixels of you, it will get the ability to attack with a sword. A sword attack will only deal damage after the "animation is finished" which will be a small timer that allows them to instantly attack and play the attack animation but only deal damage after the timer finishes allowing fair gameplay.
  - When an AI is within a set Y limit of you (currently undecided) the AI will get the ability to shoot at you. The shooting as defined by the current plan will be 100% accurate + a random inaccuracy multiple between -1 and 1 for example.
  - When an AI does not have a sword, and is within a few pixels of you, it will get the ability to hit you which will do half damage but will attack at a higher speed than a sword.
  - When an AI is staggered, it will drop any weapon it is holding allowing the player to grab it out of mid-air if time is slowed.