Technical Design Document

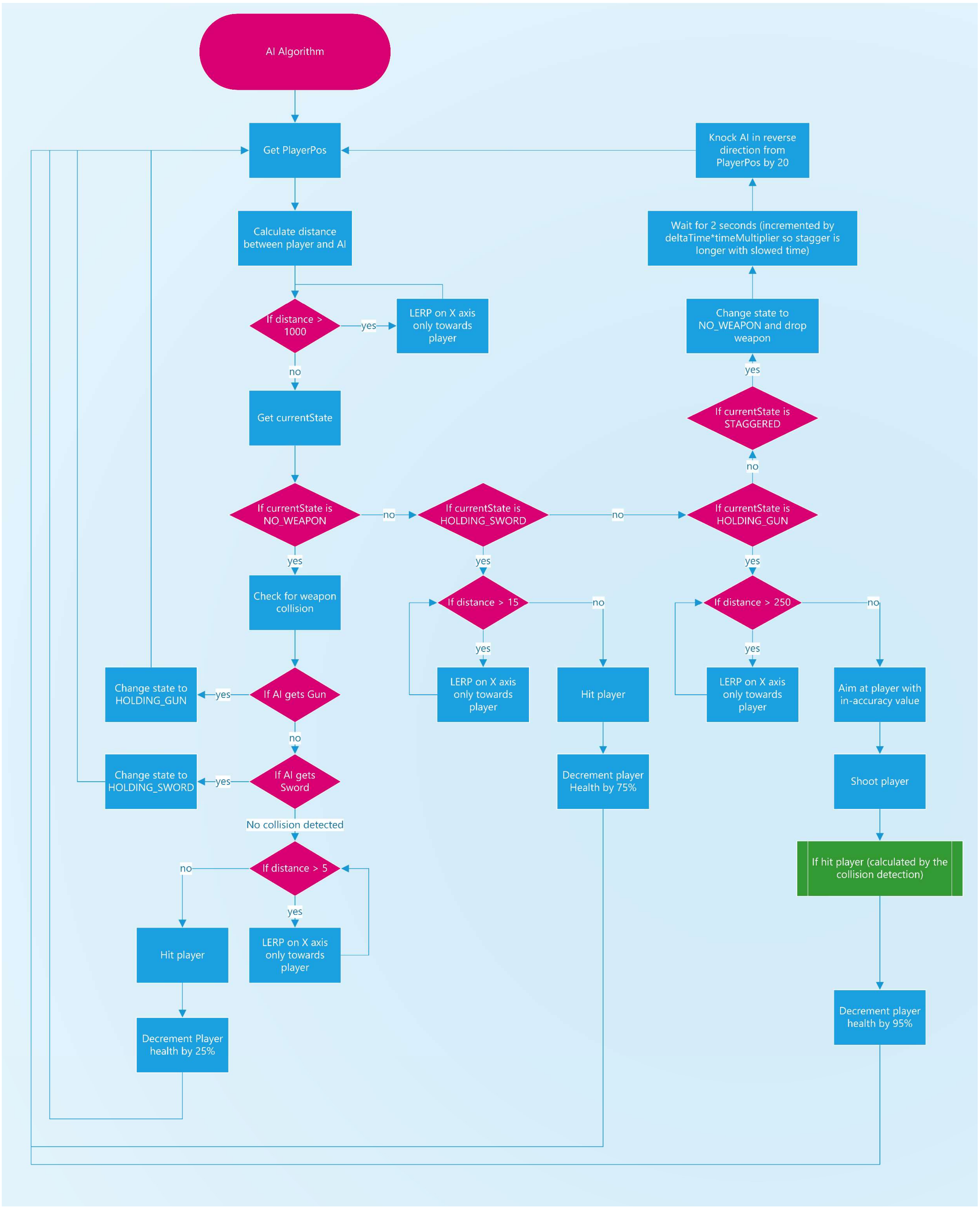
Nikkolas Diehl – 16945724
Auckland University of Technology – AUT

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

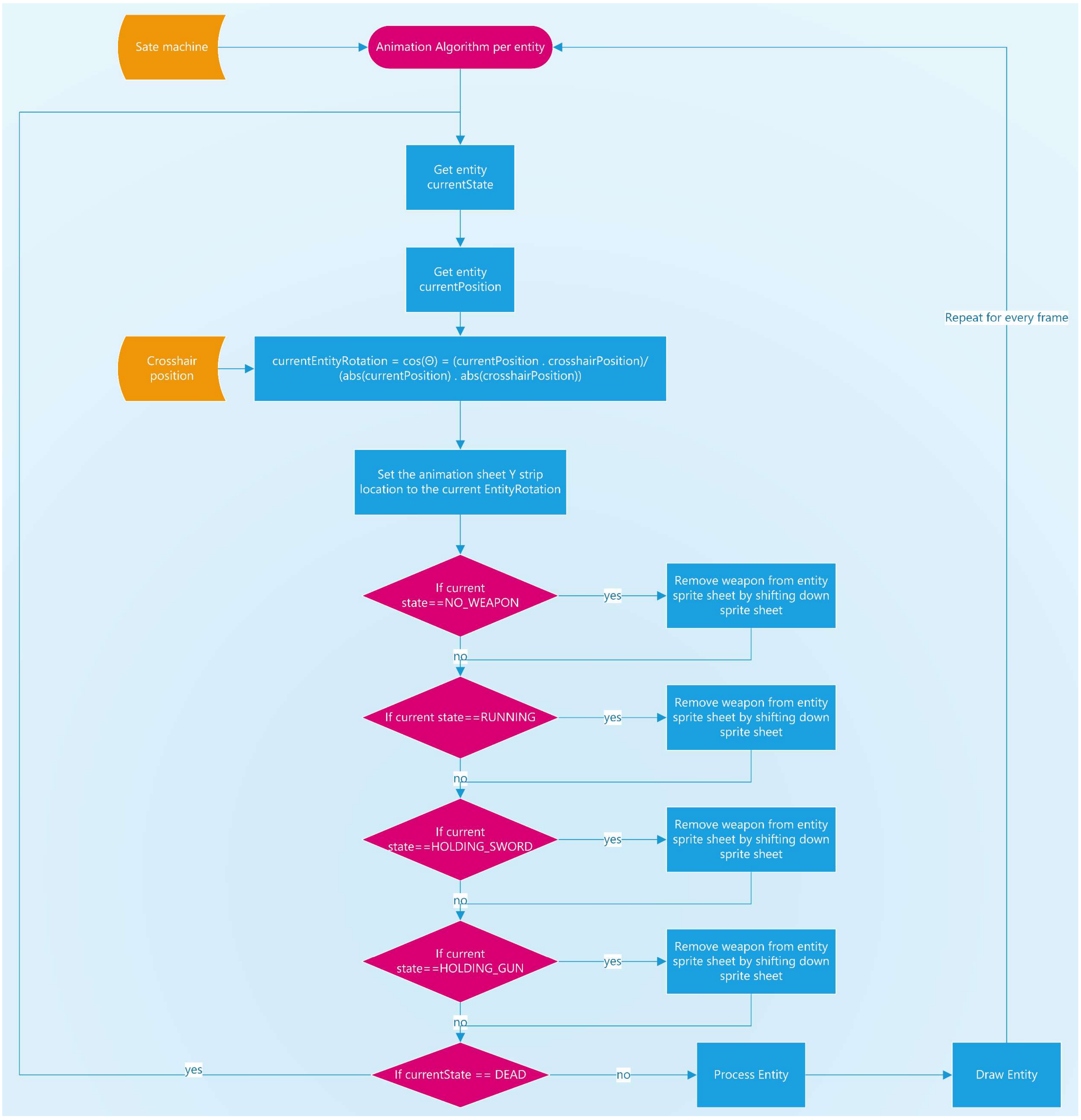
    players/AI walking through it. AI enemies that are dead will be destroyed (hidden and no longer processed/drawn) on death. Particles will spawn in place of the dead body.
    - Any thrown weapons or spawned weapons will collide with everything around it.
    - All particles, AI, players, and bullets 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.

AI ALGORITHM

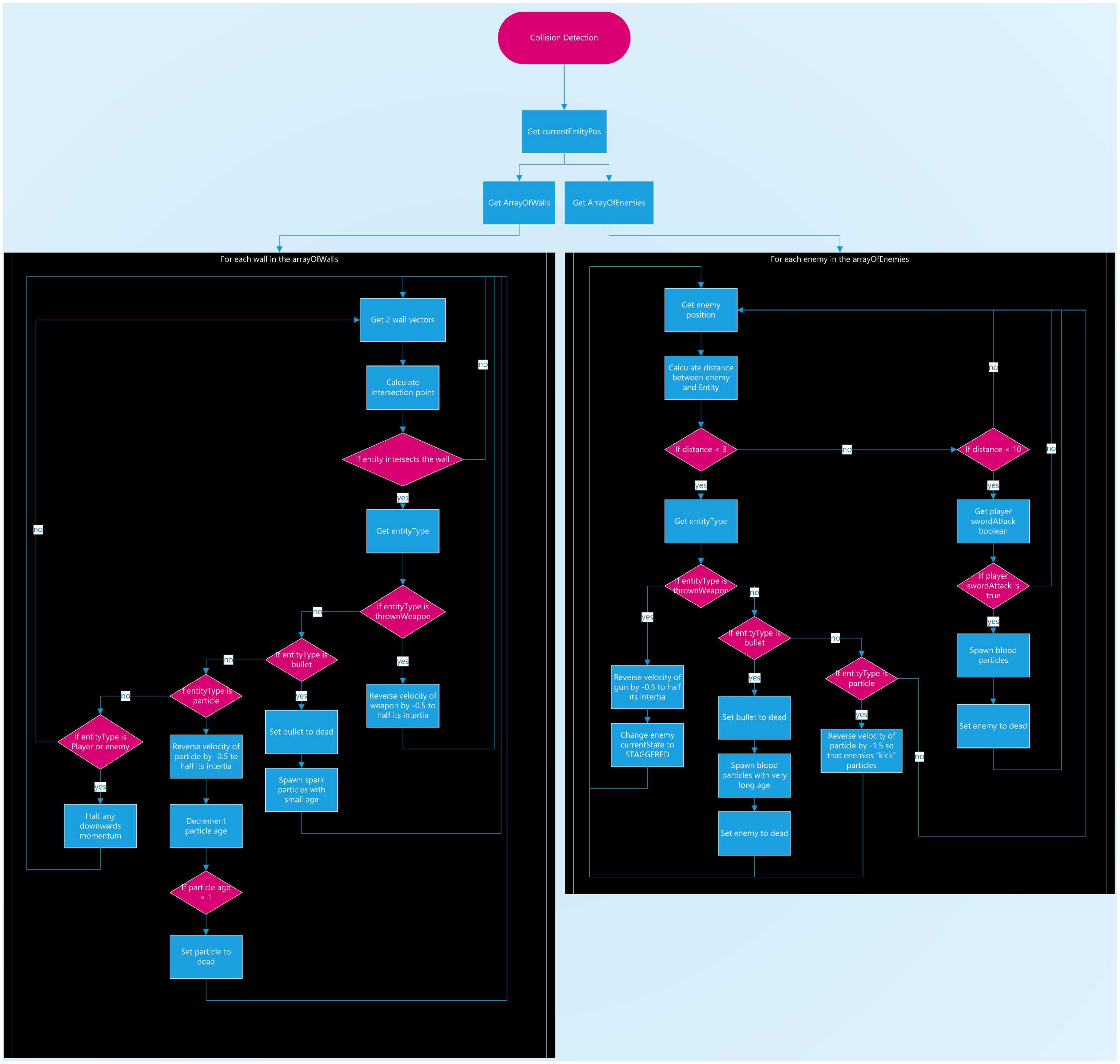


AI Algorithm
Get PlayerPos
Calculate distance between player and AI
If distance > 1000
yes
LERP on X axis only towards player
no
Get currentState
If currentState is NO_WEAPON
no
If currentState is HOLDING_SWORD
no
If currentState is HOLDING_GUN
no
If currentState is STAGGERED
yes
Change state to NO_WEAPON and drop weapon
Wait for 2 seconds (incremented by deltaTime*timeMultiplier so stagger is longer with slowed time)
Knock AI in reverse direction from PlayerPos by 20
yes
Check for weapon collision
If AI gets Gun
yes
Change state to HOLDING_GUN
no
If AI gets Sword
yes
Change state to HOLDING_SWORD
No collision detected
If distance > 5
no
Hit player
Decrement Player health by 25%
yes
LERP on X axis only towards player
yes
If distance > 15
yes
LERP on X axis only towards player
no
Hit player
Decrement player Health by 75%
yes
If distance > 250
yes
LERP on X axis only towards player
no
Aim at player with in-accuracy value
Shoot player
If hit player (calculated by the collision detection)
Decrement player health by 95%

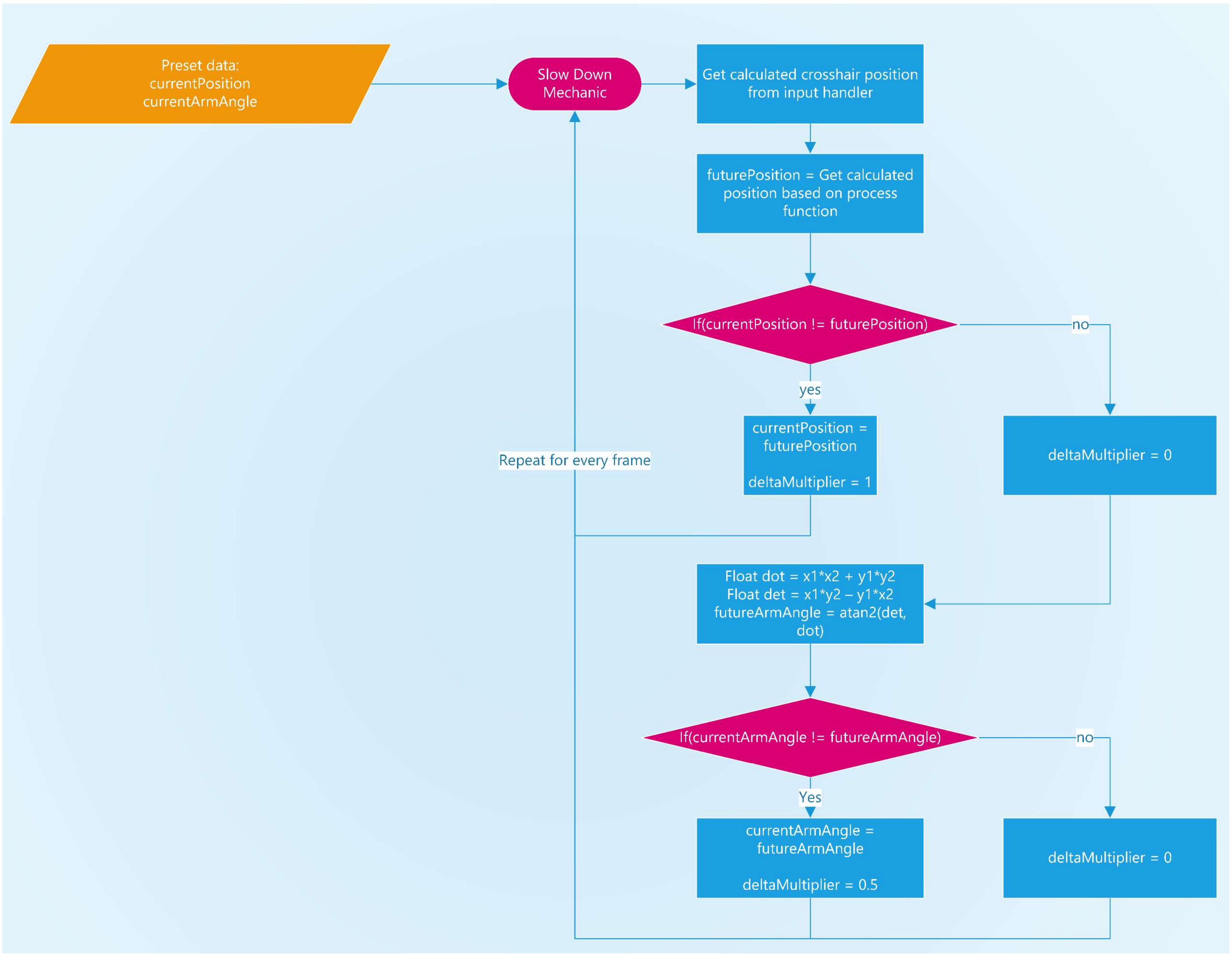
ANIMATION ALGORITHM


Sate machine
Animation Algorithm per entity
Get entity currentState
Get entity currentPosition
Crosshair position
currentEntityRotation = cos(Θ) = (currentPosition . crosshairPosition)/ (abs(currentPosition) . abs(crosshairPosition))
Set the animation sheet Y strip location to the current EntityRotation
If current state==NO_WEAPON
yes
Remove weapon from entity sprite sheet by shifting down sprite sheet
no
If current state==RUNNING
yes
Remove weapon from entity sprite sheet by shifting down sprite sheet
no
If current state==HOLDING_SWORD
yes
Remove weapon from entity sprite sheet by shifting down sprite sheet
no
If current state==HOLDING_GUN
yes
Remove weapon from entity sprite sheet by shifting down sprite sheet
no
yes
If currentState == DEAD
no
Process Entity
Draw Entity
Repeat for every frame

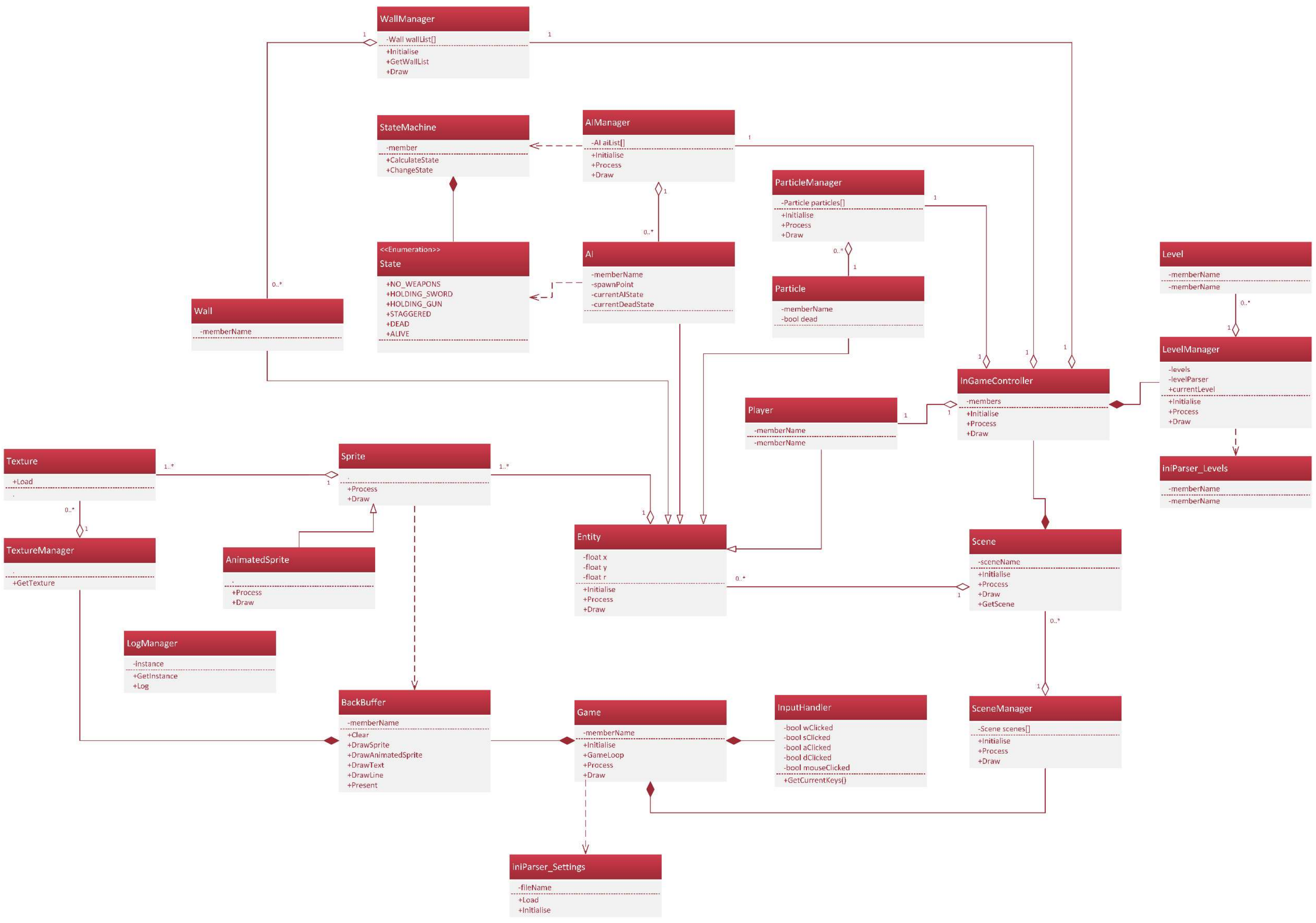
COLLISION DETECTION


Collision Detection
Get currentEntityPos
Get ArrayOfWalls
Get ArrayOfEnemies
For each wall in the arrayOfWalls
Get 2 wall vectors
Calculate intersection point
If entity intersects the wall
no
yes
Get entityType
If entityType is thrownWeapon
yes
Reverse velocity of weapon by -0.5 to half its intertia
no
If entityType is bullet
yes
Set bullet to dead
Spawn spark particles with small age
no
If entityType is particle
Reverse velocity of particle by -0.5 to half its intertia
Decrement particle age
If particle age < 1
Set particle to dead
no
If entityType is Player or enemy
yes
Halt any downwards momentum
no
For each enemy in the arrayOfEnemies
Get enemy position
Calculate distance between enemy and Entity
If distance < 3
yes
Get entityType
If entityType is thrownWeapon
yes
Reverse velocity of gun by -0.5 to half its intertia
Change enemy currentState to STAGGERED
no
If entityType is bullet
yes
Set bullet to dead
Spawn blood particles with very long age
Set enemy to dead
no
If entityType is particle
yes
Reverse velocity of particle by -1.5 so that enemies "kick" particles
no
no
If distance < 10
no
yes
Get player swordAttack boolean
If player swordAttack is true
no
yes
Spawn blood particles
Set enemy to dead

SLOW DOWN MECHANIC


Preset data:
currentPosition
currentArmAngle
Slow Down Mechanic
Get calculated crosshair position from input handler
futurePosition = Get calculated position based on process function
If(currentPosition != futurePosition)
no
yes
currentPosition = futurePosition
deltaMultiplier = 1
deltaMultiplier = 0
Repeat for every frame
Float dot = x1*x2 + y1*y2
Float det = x1*y2 – y1*x2
futureArmAngle = atan2(det, dot)
If(currentArmAngle != futureArmAngle)
no
Yes
currentArmAngle = futureArmAngle
deltaMultiplier = 0.5
deltaMultiplier = 0

UML DIAGRAM


WallManager
-Wall wallList[]
+Initialise
+GetWallList
+Draw
1
1
0..*
Wall
-memberName
StateMachine
-member
+CalculateState
+ChangeState
<<Enumeration>>
State
+NO_WEAPONS
+HOLDING_SWORD
+HOLDING_GUN
+STAGGERED
+DEAD
+ALIVE
AIManager
-AI aiList[]
+Initialise
+Process
+Draw
1
1
0..*
AI
-memberName
-spawnPoint
-currentAIState
-currentDeadState
ParticleManager
-Particle particles[]
+Initialise
+Process
+Draw
1
0..*
1
Particle
-memberName
-bool dead
Level
-memberName
-memberName
0..*
1
LevelManager
-levels
-levelParser
+currentLevel
+Initialise
+Process
+Draw
iniParser_Levels
-memberName
-memberName
1
1
1
InGameController
-members
+Initialise
+Process
+Draw
Player
-memberName
-memberName
1
1
Texture
+Load
.
1..*
1
0..*
1
TextureManager
.
+GetTexture
Sprite
.
+Process
+Draw
1..*
1
AnimatedSprite
.
+Process
+Draw
LogManager
-instance
+GetInstance
+Log
Entity
-float x
-float y
-float r
+Initialise
+Process
+Draw
0..*
1
Scene
-sceneName
+Initialise
+Process
+Draw
+GetScene
0..*
1
SceneManager
-Scene scenes[]
+Initialise
+Process
+Draw
BackBuffer
-memberName
+Clear
+DrawSprite
+DrawAnimatedSprite
+DrawText
+DrawLine
+Present
Game
-memberName
+Initialise
+GameLoop
+Process
+Draw
InputHandler
-bool wClicked
-bool sClicked
-bool aClicked
-bool dClicked
-bool mouseClicked
+GetCurrentKeys()
iniParser_Settings
-fileName
+Load
+Initialise

Nikkolas Diehl – 16945724

Relevant file Formats:

- .cpp files for code
- .h files for headers
- .ini files for settings and levels
- .png for all images and sprites
- .wav or .mp3 for all sound clips

Debug Features:

The Really Warm application/game will have a verity of different debugging features to help with the development cycle of the game. Some of the first few ones include:

- Click *Y* to kill all enemies within a level
- Click *U* to set your health to infinite
- Click *I* to fly

Naming Schemes:

The Really Warm application will feature the same/relatively similar naming scheme to the framework given to us, and the one used at Sony.

[memberState][publicState]_[type]_[variableName].

- Member state can be represented as *m* for member, *er* for external reference, *ep* for external pointer, *ec* for external copy.
- Public state can be represented as *p* for public, *x* for private, *o* for protected.
- Type can be represented as *s* for string, *I* for integer, *f* for float, *b* for Boolean, *c* for char and etc. Custom classes will not use this type naming scheme.

An example of this is a private string as a name of a *car* class would be represented as: *mx_s_name*.

Or a public floating-point number that is passed into a function as an external reference from another class could be presented as: *erp_f_variable*.

Or a protected Boolean for dead on an entity as a member variable would be represented as: *mo_b_dead*.

Acceptance test Questionnaire:

1. Does the game run to completion with no errors, warnings or memory leaks and/or crashes?
2. Does the player character respond correctly and as expected to user input for movement?
3. Is the user able to pick up weapons and swing the sword and shoot the gun?
4. Does the AI fight back against the player and try kill them and try get closer to the player character?
5. Does time slow down when the player is not moving?
6. Does time move slightly faster when moving your crosshair and even faster when the user moves the player character?

7. Does the level end/does the player win once all AI in the level are dead and the player has reached the exit?
8. Can the player and AI take damage and die?
9. Can the AI and player both pick up weapons and use them?
10. Is the game physics simulated? When particles are spawned, or the AI/player moves around, are they affected by gravity and bounce off walls correctly and as expected?

Bibliography

Christopher, Rama, & Baiherula. (n.d.). *distance between vectors*. Retrieved from varsitytutors.com: https://www.varsitytutors.com/calculus_3-help/distance-between-vectors

Line-Line intersection. (2019, August 16). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Line%E2%80%93line_intersection

MvG. (2017, December 20). *Direct way of computing clockwise angle between 2 vectors*. Retrieved from stackoverflow: https://stackoverflow.com/questions/14066933/direct-way-of-computing-clockwise-angle-between-2-vectors

Mykhailo, D. (2011). *Angle between two vectors*. Retrieved from OnlineMSchool: https://onlinemschool.com/math/library/vector/angl/