# ICS 3U Summative Project Proposal



Name: Gao, Cheng

Class: ICS3U1

Teacher: Mr. Brown

Date: May 11, 2015

A. Brief description of my game:

The game I’m going to design is called Ace Pilot. In the game, the player will have an adventure in the sky with as an airplane. The player has to destroy all the enemies and a final boss so that the player can get into the next stage and continue to play. A small amount of gold will be rewarded for each enemy the player has destroyed; massive gold will be reward when the player kills the boss. Each time the player clean a stage and each stage have 10 enemies and 1 final boss. The player is able to use the gold that earned from the previous stage to update one of the air plane’s abilities. The score will keep updating while the game is running. The player will lose when armor becomes 0 or win the game when finish stage 10

B. A diagram that shows what my game window will look like:

This is the player’s current lives.

Each time the player’s armor becomes 0, the current lives will be decreased by 1.

This is where it shows the airplane’s remaining armor. Every time the player collides with the bullet or the enemy, the player’s armor will be reduced. The game will be closed when the armor becomes.

This is where it shows the player’s current score.The player’s score will be added when the player destroys a enemy or the boss of one stage

This will indicate the current stage the player is in

This block is player’s missiles. It will decrease enemies’ health if it collide with the enemies.

These blocks are enemies. It will decrease the player’s armor if enemies collide with the player or it will kill the enemies if the player’s missiles collide with the enemies

This is the final boss of a stage. Its health is way more than the normal enemies. Therefore, more gold the player will receive after destroying it. The boss will cast missiles or release its ultimate shot every certain period of time

Apparently, this will be the player! The player can cast bullets to destroy the enemies and update their aircraft every time they clean a stage

This is the ability update surface:

This will indicate how much gold the player needs to update the armor

This will indicate how much gold the player needs to update the speed

This box will load an image of armor to hint the player this is where they update the airplane’s armor

This box will load an image of missile to hint the player this is where they update the airplane’s powder

This box will load an image of boots to hint the player this is where they update the airplane’s speed

This will indicate how much gold the player needs to update the missile power

C. A detailed description of each custom sprite class:

class Player():

\_\_init\_\_(): #initialize the attributes of the sprite

self.image #an picture that represents the airplane

self.\_\_screen # to keep track of the screen surface and it will be needed in other methods

self.rect

self.rect.center #put the player at the bottom center of the screen

change\_direction(x-speed, y-speed): #assign x,y speed for player

current\_position(): #return the player’s current left and top sides

update(): #reposition the sprite

self.rect.left #the player’s left side changes based on the x-speed

self.rect.top #the player’s top side changes based on the y-speed

class PlayerMissile():

\_\_init\_\_(inceased\_missle\_power): #initialize the attributes of the sprite

self.image #an picture that represents the bullet

self.\_\_dy #y distance that the bullet moves

self.\_\_screen # to keep track of the screen surface and it will be needed in other methods

self.\_\_missile\_power #an integer that represents the power of the bullet

self.\_\_missile\_power += increased\_missle\_power

current\_missile\_power(): #return current missile power

follow\_player(player\_current\_position): #this method ensures that the missile’s center is at the player’s top-center

self.rect.left, self.rect.top = player\_current\_position

update(): #reposition the sprite

class Enemy():

\_\_init\_\_(increased\_health): #initialize the attributes of the sprite

self.image #an picture that represents the enemy

self.\_\_dx #x distance that the enemy moves

self.\_\_dy #y distance that the enemy moves

self.\_\_screen #to keep track of the screen surface and it will be needed in other methods

self.rect

self.rect.center #generate a random integer for x-position, y-position will be the height of the screen

self.\_\_health #an integer that represents the health of the enemy. The enemy will be destroyed when the health is decreased to 0

self.\_\_health += increased\_health #the health of the enemies will increase as the stage number increases

current\_health(): #return the current health of the enemy

lose\_health(decreased\_value):

self.\_\_health -= decreased\_value

current\_position(): #return the enemy’s current left and top sides

update(): #reposition the sprite

class EnemyMissile():

\_\_init\_\_(inceased\_missle\_power): #initialize the attributes of the sprite

self.image #an picture that represents the bullet

self.\_\_dy #y distance that the bullet moves

self.\_\_screen # to keep track of the screen surface and it will be needed in other methods

self.\_\_missile\_power #an integer that represents the power of the bullet

self.\_\_missile\_power += increased\_missle\_power

follow\_enemy(enemy\_current\_position): #this method ensures that the missile’s center is at the player’s top-center

self.rect.left, self.rect.top = enemy\_current\_position

update(): #reposition the sprite

class Boss():

\_\_init\_\_(screen, increased\_health): #initialize the attributes of the sprite

self.image #load an image of the boss

self.\_\_dx #x distance that the boss moves

self.\_\_screen #to keep track of the screen surface and it will be needed in other

methods

self.rect

self.rect.center

self.\_\_health #an integer that represent the boss’s health

self.\_\_health += increased\_health #increase boss’s health

current\_health(): #return the boss’s current health

lose\_health(decreased\_value):

self.\_\_health -= decreased\_value

current\_position(): #return the boss’s current left and top sides

update(): #reposition the sprite

class BossMissile():

\_\_init\_\_(inceased\_missle\_power): #initialize the attributes of the sprite

self.image #an picture that represents the bullet

self.\_\_dy #y distance that the bullet moves

self.\_\_screen # to keep track of the screen surface and it will be needed in other methods

self.\_\_missile\_power #an integer that represents the power of the bullet

self.\_\_missile\_power += increased\_missle\_power

follow\_boss(boss\_current\_position): #this method ensures that the missile’s center is at the player’s top-center

self.rect.left, self.rect.top = enemy\_current\_position

update(): #reposition the sprite

class ScoreKeeper():

\_\_init\_\_(screen): #initialize the attributes of the sprite

self.\_\_font #load a custom font

self.\_\_player\_score #an integer that indicates the player’s current score, starting with 0

player\_scored(): #increase the current score

update(): #This method will be called automatically to display

the current score at the game window.

message #a string that indicates the player’s current score in text

self.image #a label that displays the message in the font that we set

self.rect

self.rect.center

class StageCounter():

\_\_init\_\_(screen,stage\_number): #initialize the attributes of the sprite

self.\_\_stage = stage\_number #an integer that indicates the current stage that the player is in

self.\_\_font #load a custom font

current\_stage(): #return current stage number

update(): #This method will be called automatically to display

the current stage at game window.

message #a string that indicates the current stage in text

self.image #a label that displays the message in the font that we set

self.rect

self.rect.center

class Lives():

\_\_init\_\_(screen): #initialize the attributes of the sprite

self.\_\_lives #an integer that indicates the player’s lives

self.\_\_font #load a custom font

lose\_lives(decreased\_value):

self.\_\_lives -= decreased\_value

current\_lives(): #an integer that indicates the player’s current lives

update(): #This method will be called automatically to display

the current lives at the game window.

message #a string that indicates the player’s remaining lives in text

self.image #a label that displays the message in the font that we set

self.rect

self.rect.center

class Armor(): #this method takes a parameter as the value of the increased armor

\_\_init\_\_(screen, initial\_armor, new\_armor): #initialize the attributes of the sprite

self.\_\_armor = initial\_armor + new\_armor #increase the player’s armor

self.\_\_font #load a custom font

armor\_decrease(decreased\_value):

self.\_\_armor -= decreased\_value

current\_armor(): #return current armor

update(): #This method will be called automatically to display

the current armor at game window.

message #a string that indicates the current armor in text

self.image #a label that displays the message in the font that we set

self.rect

self.rect.center

class UpdateMissle():

\_\_init\_\_(screen):

self.image #load an image of missle

self.\_\_font #load a custom font

self.\_\_cost #fee needed to update the missiles

update(): #this method will be called automatically to display

the fee at game window.

message #a string that indicates the fee in text

self.image #a label that displays the message in the font that we set

self.rect

self.rect.center

class UpdateArmor():

\_\_init\_\_(screen):

self.image #load an image of armor

self.\_\_font #load a custom font

self.\_\_cost #fee needed to update the armor

update(): #this method will be called automatically to display

the fee at game window.

message

self.image #a label that displays the message in the font that we set

self.rect

self.rect.center

class UpdateSpeed():

\_\_init\_\_(screen):

self.image #load an image of boots

self.\_\_font #load a custom font

self.\_\_cost #fee needed to update the armor

update(): #this method will be called automatically to display

the fee at game window.

message

self.image #a label that displays the message in the font that we set

self.rect

self.rect.center

D. Step lists to implement the game:

1. Create sprite classes for the game
2. Import pygame and the Sprite document(called mySprites for now)
3. load a game window
4. load the player at the bottom center of the screen
5. The player will use the arrow keys and space to control the airplane
6. load scorekeeper, armor, lives and stage number on the top of the screen
7. The enemies will be loaded on the top of the screen with a random x-position every certain seconds
8. Every time the player’s missile hit an enemy, it will decrease the enemy’s health. The enemy will disappear of its health becomes 0
9. Same logic with the boss
10. Same logic with the player. Every time the player’s armor becomes 0, its remaining live will be decreased by 1
11. Add some special effect when the enemy, boss or the player aircraft is destroyed
12. Add background sound
13. Add sound effect
14. If all the enemies and the boss on the stage are cleared, a update interface will be loaded
15. The player can choose to update one of the aircraft’s abilities
16. After the player update its airplane’s ability, then initialize all the sprites with new value(ex. The stage number will be increased by 1)
17. If all the 10 stages are cleared or the player’s remaining lives becomes 0, load a “GAME OVER” image on the screen, and then the game will be closed