Lokaskýrsla

# **HBV402G – Vor 2018**

Verkefnahópur 1:

Oddný Huld Halldórsdóttir

Stefán Þór Þorgeirsson

Sæþór Tryggvasson

Vésteinn Tryggvasson

Apríl 15, 2018

Þetta er viðmiðunarskjal fyrir hópavinnu við lok allra spretta. Markmiðið er að hanna tæknilegt skjal með nægum upplýsingum til að túlka hugbúnaðarvöruna og áætlaðri vinnu fyrir næstu skref samtvinnað með útskýrungum hvað hefur áunnist. Það má einnig stokka upp vissa kafla t.d gæti kafli 2 samanstaðið af ytri bókasöfnum (libraries) og eiginn hönnun. Skjalið má ver á bilinu 4 til 10 síður.

## **Inngangur**

Þetta verkefni snýr að hönnun á grafískum sýndarleik sem byggist á textaleiðbeiningum. Leikurinn lýsir sér þannig að leikmaður byrjar í fyrsta heimi leikjar og á að ferðast á milli heima. Leikurinn, Lunartic, gerist á yfirgefinni tunglstöð og er það markmið leikmanns að komast að því hvað varð stöðinni að falli. Fyrsti heimur er lendingarpallur eldflauga til og frá stöðinni og þar getur leikmaður byrjað að leita vísbendinga.

Verkefnið er unnið í fjórum sprettum sem hver og einn endar á skilum verkáætlunar og stöðuskýrslu sem og kóða leiks.

Upphafleg markmið verkefnis voru að uppfylla eftirfarandi skilyrði um virkni leiks:

1. Karakter með skilgreiningu (nafn, kyn, lýsing, etc.)
2. Íhlutir með sérstökum einkennum og eiginleikum (nafn, lýsing og tegund)
3. Herbergi með sérstökum eiginleikum og einkennum (nafn, lýsing, tenging við önnur herbergi)
4. Möguleikar að ferðast milli herbergja og hafa samskipti við aðrar einingar leiksins

Í Lunartic fær leikmaður tækifæri til að velja milli þriggja karaktera. Hægt er að sjá styrkleika og veikleika hvers og eins karakters og hafa þeir áhrif á samskipti við einingar.

Leikmaður hefur leik í fyrsta heimi leikjar en þeir eru þrír. Fyrsti heimur er lendingarpallur eldflauga til og frá tunglstöð, sem er aðalumhverfi leiks, og þar getur leikmaður byrjað að leita vísbendinga.

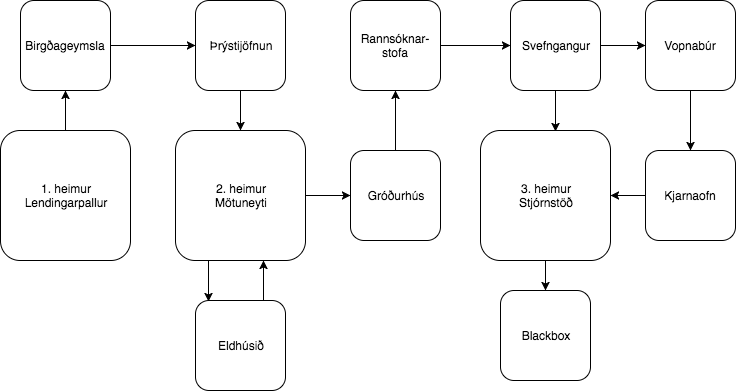
Annar heimur er mötuneyti stöðvarinnar en þar hittir leikmaður eftirlifendur og þarf að koma þeim til hjálpar eða verja sjálfan sig.

Þriðji heimurinn er síðan stjórnstöð stöðvarinnar en þar kemst leikmaður að því hvað gerðist í tunglstöðinni.

Hver heimur hefur síðan herbergi sem leikmaður getur farið inn og út úr.

Leikmaður tapar ef hann læsist inni í herbergi, rennur út á tíma eða lætur lífið.

Leikmaður vinnur ef hann kemst að svarta kassanum (e. blackbox) inni í stjórnstöð stöðvar.



Mynd 1: Kerfismynd

Hönnun leiks var að mestu leiti byggð á kerfismyndinni að ofan. Hins vegar kom í ljós undir lok að umfang hönnunar var meira en búist var við í fyrstu og því fylgir lokaútkoman ekki alveg skipulaginu að ofan.

### 

## **Hönnun, hugbúnaðar- kerfi og strúktúr**

### · **Notkun á ytri einingu, t.d. (libraries)**

Notast var við hugbúnað sem heitir Pygame við þróun leiksins.

### · **Skrár**

Notast var við nokkrar tvær skrár með myndum til að teikna umhverfi leiksins, einnig er önnur skrá sem heitir constants.py , hún heldur utan um fastar breytur í leiknum.

### · **Klasar**

### **Sjá neðar í skýrslu.**

### 

## **Núverandi staða verkefnisins**

Hér fyrir neðan má sjá þá þá þætti sem eru loknir í verkefninu:

* Byrjað var að gera greiningu (norendasögur og tasks)
* Notendasögur voru uppfærðar fyrir hvern sprett
* Búið er að gera UML rit fyrir kóðann
* Verkáætlun uppfærð í hverjum spretti
* Stöðuskýrsla uppfærð í hverjum spretti
* Kostnaðaráætlun gerð og uppfærð í hverjum spretti
* Búið að semja sögu leiks
* Forritun:
  + Búið til klasarit fyrir leikjaborð
  + Búa til klasarit fyrir leikmenn og gera þeim kleyft að hreyfast
  + Búið er að gera visual map með einingum
* Prófanir (unittests)
* Áhættumat

*Virkni og hvað hefur verið gert?*

Lokaskilin er leikur þar sem búið er að búa til karakter sem getur hreyft sig til hliðar og fram og aftur. Hann hreyfir sig á korti og getur farið fram hvert sem hann vill nema það séu hindranir í kringum hann. Kortið er gert úr einingum og er búið að setja inn myndir fyrir hverja einingu.

Leikurinn hefur einnig karaktera sem eru stjórnaðir af tölvunni, þeir geta hreyft sig eftir mapinu og ráðist á leikmanninn. Leikmaður getur ráðist á þá og drepið, en þeir geta ekki drepið hann.

Eitt af því sem er lokið við gerð leiksins er að búa til sögu leiksins en hana má sjá í viðauka. Inn í þeirri skrá má einnig sjá textaskipanir sem átti eftir að setja inn í leikinn.

*Gert hefur verið í kringum leikinn.*

Verkefnið var unnið í sprettum þar sem í enda hvers spretts var stöðuskýrslu og verkefnaáætlun skilað. Í stöðuskýrslunni voru verkþættir sem voru loknir og verkþættir sem náðist ekki að klára listaðir ásamt stutta umfjöllun um útgáfu verkefninsins í heild. Verkefnaáætlunin var uppfærð og bætt við köflum í hverjum spretti en í henni má meðal annrs finna kostnaðaráætlun og hver áætlaður tímafjöldi var settur í verkefnið. Hann var svo borinn saman við rauntímann sem fór í verkefnið. Í verkáætluninni má einnig finna grunnupplýsingar um verkefnið, eins og markmið og áætlun en einnig má finna í viðauka áhættumat og leikjasögu.

### · **Mögulegar villur (bug list)**

### · **Verkþættir sem vantar (missing features)**

Eftirfarandi þætti tókst ekki að gera vegna tíma og umfang verkþátta:

* Láta textaskipanir virka
* Geta unnið leikinn

### 

## **Næstu skref**

### · **Hvaða hönnunarframkvæmdir koma til í næsta sprint(s)**

Næstu skref í verkefninu væru þá að klára verkþættina sem ekki tókst að gera og mögulega að vinna meira í grafík til að fá flottara útlit á leikinn.

### 

### · **Möguleg langtímamarkmið (gróflega skilgreind)**

Möguleg langtímamarkmið væru að hanna betri klasa fyrir hegðun óvinar, eins og er stjórnast þeir algjörlega að fallinum random, þannig þeir taka ekkert sérstaklega góðar ákvarðanir.

Einnig væri gaman að gera einhverskonar kerfi þannig leikmaður geti tekið upp hluti og sett á sig til að bæta styrk karaktersins. Það vantar main menu á leikinn og einnig vantar einhverja leið til að velja mismunandi karaktera til að spila. Allt væri hægt að bæta við í framtíðinni

## **Hvað hefur lærst og áunnist**

Frekar mikið stökk í djúpulaugina þegar það kemur að forritun, en á sama tíma mjög skemmtileg reynsla.

Fengum góða innsýn inn í hversu flókin hugbúnaðar verkefni geta verið í þróun og hönnun. Kom sérstaklega í bakið á okkur að hafa aðeins tvo að kóða eftir sprett 2. Því það var eiginlega ógerlegt að bæta við kóðurum seinna, án þess að þurfa eyða miklum tíma í að útskýra það sem komið var.

### **Samantekt**

Lunartic var metnaðarfullt og lærdómsríkt verkefni. Það sem eftir situr er hversu umfangsmikil hugbúnaðarþróun getur verið og hvernig mismunandi aðilar geta komið að mismunandi verkefnum innan fagsins. Við verkefnaskil hefur tekist að mæta flestum grunndvallarkröfum sem lagt var upp með í byrjun. Útkoman er keyranlegur leikur með grafík og einingum sem gæti farið í frekari þróun í framhaldinu. Leikurinn er byggður á grunnhugmyndinni að baki Lunartic þar sem mismunandi herbergi tunglstöðvar er umhverfi leikspilunar. Ekki náðist að fá textaskipanir til að virka þó að þær séu til staðar (sjá leikjasögu í viðauka verkáætlunar 4). Þá náðist ekki að gera leikmanni kleift að vinna. Ljóst er að áframhaldandi þróun á leiknum mundi tækla þessi vandamál sem og að vinna að bættri grafík til að gera umhverfið meira spennandi.

## 

## 

## 

## 

## 

## 

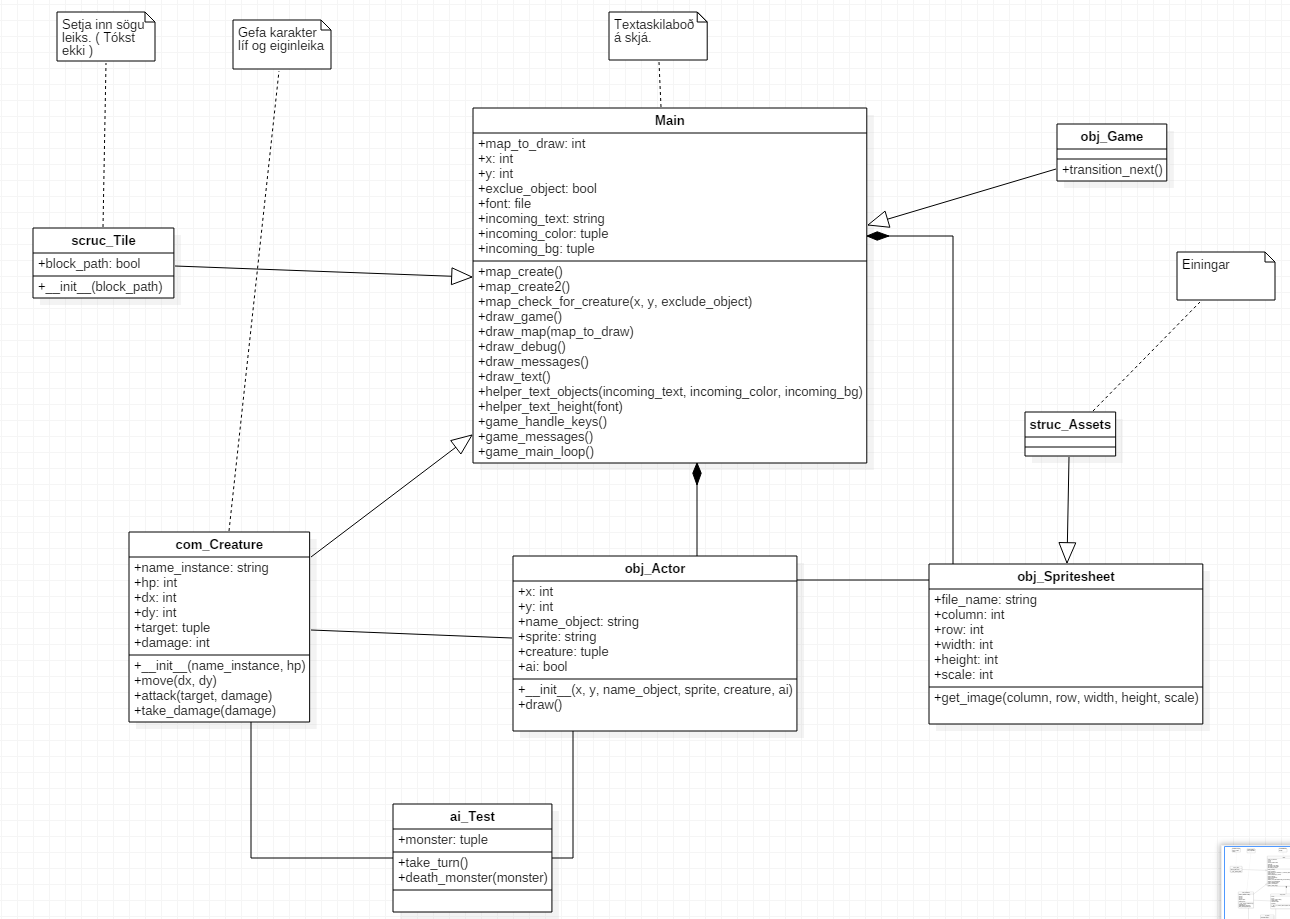
## 

## 

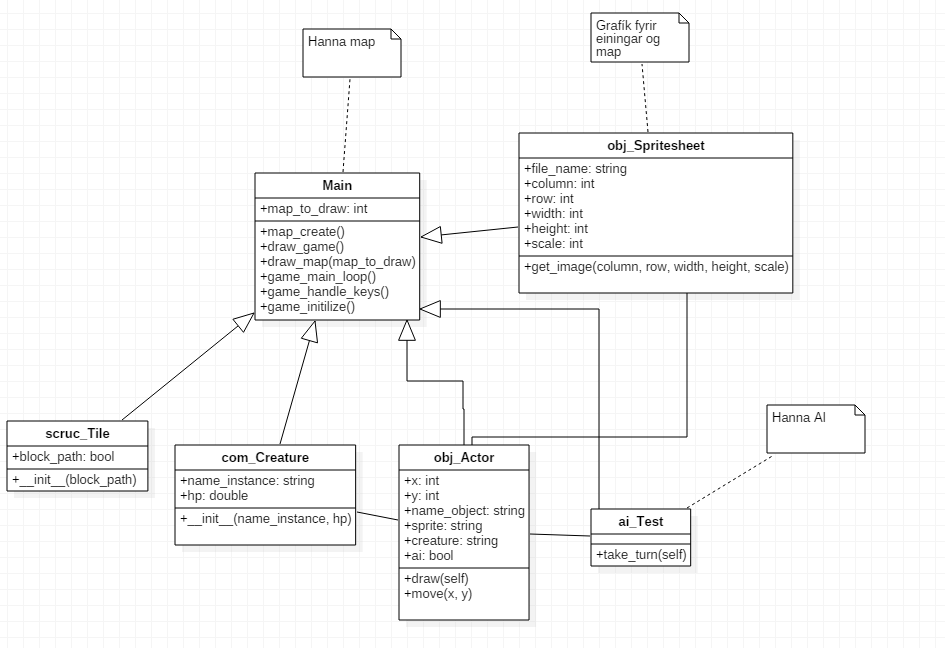
## 

## **Klasa diagröm**

Sprint 4.



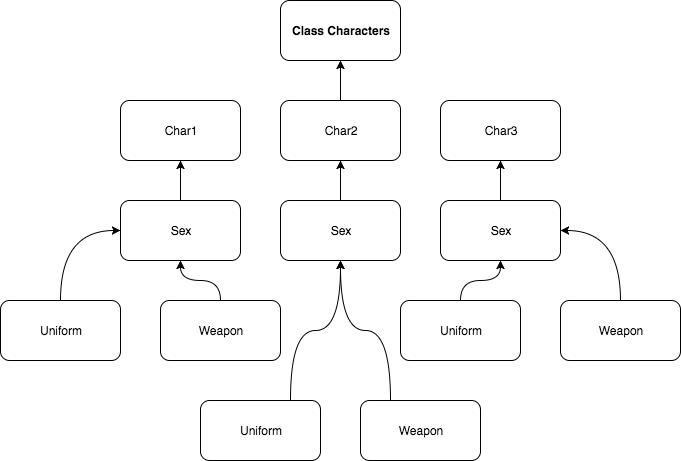
Sprint 3.



Sprint 2:

## 

Sprint 1:

****

## 

## 

## **Forritunarkóði**

# 3rd party modules and python libraries

import pygame

import random

import itertools as it

# game files

import constants

#############################################

# Structure

#############################################

class struc\_Tile:

# if tile is False player can walk over it.

# if tile is True player can NOT walk over it.

def \_\_init\_\_(self, block\_path):

self.block\_path = block\_path

class struc\_Assets:

def \_\_init\_\_(self):

# Spritesheet

self.playerspritesheet = obj\_Spritesheet('data/char.png')

self.charspritesheet = obj\_Spritesheet('data/alien.png')

# Sprites ( Column, row, sprite size m / n , scale size )

self.S\_PLAYER = self.playerspritesheet.get\_image('b', 4, 32, 32, (32, 32))

self.S\_ENEMY = self.charspritesheet.get\_image('g', 8, 32, 32, (32, 32))

self.S\_ENEMY2 = self.charspritesheet.get\_image('g', 12, 32, 32, (32, 32))

self.S\_WALL = self.charspritesheet.get\_image('i', 0, 32, 32, (32, 32))

self.S\_FLOOR = self.charspritesheet.get\_image('k', 4, 32, 32, (32, 32))

# Fonts (Font file, size of font)

self.FONT\_DEBUG\_MESSAGE = pygame.font.Font('data/joystix.ttf', 20)

self.FONT\_MESSAGE\_TEXT = pygame.font.Font('data/joystix.ttf', 20)

#############################################

# Components

#############################################

class com\_Creature:

# Creatures have health and can damage other objects by attacking them,

# can also die.

def \_\_init\_\_(self, name\_instance, hp = 10, death\_function = None):

self.name\_instance = name\_instance

self.maxhp = hp

self.hp = hp

self.death\_function = death\_function

# Move the actor. dx = distance to move x. dy = distance to move y.

def move(self, dx, dy):

# The actor checks where it and where he wants to move to.

tile\_is\_wall = (GAME.current\_map[self.owner.x + dx][self.owner.y + dy].block\_path == True)

# Checks if creatures are in x or y direction, also checks self

# since we don't want creatures to hurt themself if they don't move.

target = map\_check\_for\_creature(self.owner.x + dx, self.owner.y + dy, self.owner)

if target:

self.attack(target, 5)

if not tile\_is\_wall and target is None:

self.owner.x += dx

self.owner.y += dy

# Allows creatures to dealdamage.

def attack(self, target, damage):

game\_message(self.name\_instance + " attacks " +

target.creature.name\_instance + " for " +

str(damage) + " damage!", constants.COLOR\_WHITE)

target.creature.take\_damage(damage)

# Allows creatures to take damage.

def take\_damage(self, damage):

self.hp -= damage

game\_message(self.name\_instance + "'s health is " +

str(self.hp) + "/" + str(self.maxhp), constants.COLOR\_RED)

# Checks if creature hp is 0 or lower.

if self.hp <= 0:

if self.death\_function is not None:

self.death\_function(self.owner)

#############################################

# AI

#############################################

class ai\_Test:

# Once per turn, exectue.

def take\_turn(self):

self.owner.creature.move(random.randint(-1,1), random.randint(-1,1))

def death\_monster(monster):

# On death, makes monsters stop moving

game\_message(monster.creature.name\_instance + " is dead!",

constants.COLOR\_GREY)

# Turns off creature component and ai

monster.creature = None

monster.ai = None

#############################################

# Objects

#############################################

class obj\_Actor:

# Our basic actor object.

def \_\_init\_\_(self, x, y, name\_object, sprite, creature = None, ai = None):

# Create the actor and set its coordinates.

self.x = x # map address (not a pixel address)

self.y = y # map address (not a pixel address)

self.sprite = sprite

self.creature = creature

if creature:

self.creature = creature

creature.owner = self

self.ai = ai

if ai:

ai.owner = self

def draw(self):

# Function to have the actor draw itself.

SURFACE\_MAIN.blit(self.sprite, (self.x\*constants.CELL\_WIDTH,

self.y\*constants.CELL\_HEIGHT))

class obj\_Game:

# Swapping out global variables for objects.

def \_\_init\_\_(self):

self.current\_map = map\_create()

self.current\_objects = []

self.message\_history = []

def transition\_next(self):

self.current\_map = map\_create2()

self.current\_objects = [PLAYER, ENEMY2]

#############################################

# Sprites

#############################################

class obj\_Spritesheet:

# Class used to grab images out of a sprite sheet

def \_\_init\_\_(self, file\_name):

# Load the sprite sheet.

self.sprite\_sheet = pygame.image.load(file\_name).convert()

self.tiledict = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5,

'f': 6, 'g': 7, 'h': 8,'i': 9,'j' :10,'k': 11,'l': 12

,'m': 13,'n': 14,'o': 15,'p': 16}

def get\_image(self, column, row, width = constants.CELL\_WIDTH,

height = constants.CELL\_HEIGHT, scale = None):

# Scale is a tuple, returns a single image

image = pygame.Surface([width, height]).convert()

# Blit the image to assigned location, last two rows are the image adress in sprite sheet

image.blit(self.sprite\_sheet, (0, 0), (self.tiledict[column]\*width,

row\*height, width, height))

image.set\_colorkey(constants.COLOR\_BLACK)

# Scaling the images if we need to scale it

if scale:

(new\_w, new\_h) = scale

image = pygame.transform.scale(image, (new\_w, new\_h))

return image

#############################################h

# Map

#############################################

# Did not get the map editor to work properly, had to hardcore the map

# Needs a fix, this is not proper coding... temp fix with itertools.

def map\_create():

new\_map = [[struc\_Tile(False) for y in range(0, constants.MAP\_HEIGHT)] for x in range(0, constants.MAP\_WIDTH)]

# map borders x direction

for x in range(constants.MAP\_WIDTH):

new\_map[x][0].block\_path = True

new\_map[x][constants.MAP\_HEIGHT-1].block\_path = True

# map borders y direction

for y in range(constants.MAP\_HEIGHT):

new\_map[0][y].block\_path = True

new\_map[constants.MAP\_WIDTH-1][y].block\_path = True

for x in range(11, 18):

new\_map[x][6].block\_path = True

new\_map[x][constants.MAP\_HEIGHT-1].block\_path = True

for x in range(11, 18):

new\_map[x][3].block\_path = True

new\_map[x][constants.MAP\_HEIGHT-1].block\_path = True

for x in it.chain(range(1, 21), range(23, constants.MAP\_HEIGHT)):

new\_map[x][9].block\_path = True

new\_map[x][constants.MAP\_HEIGHT-1].block\_path = True

for x in it.chain(range(1, 4), range(7, 13),

range(16, constants.MAP\_HEIGHT)):

new\_map[x][18].block\_path = True

new\_map[x][constants.MAP\_HEIGHT-1].block\_path = True

for y in it.chain(range(1, 4), range(6, 9),

range(19, constants.MAP\_HEIGHT)):

new\_map[10][y].block\_path = True

new\_map[constants.MAP\_WIDTH-1][y].block\_path = True

for y in it.chain(range(1, 4), range(6, 12), range(15, 21),

range(23, constants.MAP\_HEIGHT)):

new\_map[18][y].block\_path = True

new\_map[constants.MAP\_WIDTH-1][y].block\_path = True

return new\_map

def map\_create2():

new\_map = [[struc\_Tile(False) for y in range(0, constants.MAP\_HEIGHT)] for x in range(0, constants.MAP\_WIDTH)]

# map borders x direction

for x in range(constants.MAP\_WIDTH):

new\_map[x][0].block\_path = True

new\_map[x][constants.MAP\_HEIGHT-1].block\_path = True

# map borders y direction

for y in range(constants.MAP\_HEIGHT):

new\_map[0][y].block\_path = True

new\_map[constants.MAP\_WIDTH-1][y].block\_path = True

return new\_map

def map\_check\_for\_creature(x, y, exclude\_object = None):

target = None

# check objectlist to find creature at that location that isn't excluded.

if exclude\_object:

for object in GAME.current\_objects:

if (object is not exclude\_object and

object.x == x and

object.y == y and

object.creature):

target = object

if target:

return target

else:

# check objectlist to find any creature at that location.

for object in GAME.current\_objects:

if (object.x == x and

object.y == y and

object.creature):

target = object

if target:

return target

#############################################

# Drawing

#############################################

def draw\_game():

global SURFACE\_MAIN

# clear the surface

SURFACE\_MAIN.fill(constants.COLOR\_DEFAULT\_BG)

# draw the map

draw\_map(GAME.current\_map)

#draw the character

for obj in GAME.current\_objects:

obj.draw()

draw\_debug()

draw\_messages()

# update the display

pygame.display.flip()

def draw\_map(map\_to\_draw):

for x in range(0, constants.MAP\_WIDTH):

for y in range(0, constants.MAP\_HEIGHT):

if map\_to\_draw[x][y].block\_path == True: # if True there is a wall here

#draw wall

SURFACE\_MAIN.blit(ASSETS.S\_WALL, (

x\*constants.CELL\_WIDTH, y\*constants.CELL\_HEIGHT))

else:

#draw floor

SURFACE\_MAIN.blit(ASSETS.S\_FLOOR, (

x\*constants.CELL\_WIDTH, y\*constants.CELL\_HEIGHT))

def draw\_debug():

# Tracks fps

draw\_text(SURFACE\_MAIN, "fps: " + str(int(CLOCK.get\_fps())),

(0, 0), constants.COLOR\_WHITE, constants.COLOR\_BLACK)

def draw\_messages():

# Draws the items on the list. Negative to draw last messages

if len(GAME.message\_history) <= constants.NUM\_MESSAGES:

to\_draw = GAME.message\_history

else:

to\_draw = GAME.message\_history[-constants.NUM\_MESSAGES:]

text\_height = helper\_text\_height(ASSETS.FONT\_MESSAGE\_TEXT)

# Equal to the height of the window, minus number of messages times text height

start\_y = constants.MAP\_HEIGHT\*constants.CELL\_HEIGHT- (constants.NUM\_MESSAGES \* text\_height) - 5

i = 0

for message, color in to\_draw:

draw\_text(SURFACE\_MAIN, message, (0, start\_y + (i \* text\_height)),

color, constants.COLOR\_BLACK)

i += 1

# Takes in text and displays on the refrenced surface

def draw\_text(display\_surface, text\_to\_display,

T\_coords, text\_color, back\_color = None):

# Get both the surface and rectangle of the desired message

text\_surf, text\_rect = helper\_text\_objects(text\_to\_display, text\_color, back\_color)

# adjust the location of the surface based on the coordinates

text\_rect.topleft = T\_coords

# draw the text onto the display surface

display\_surface.blit(text\_surf, text\_rect)

#############################################

# Helpers

#############################################

def helper\_text\_objects(incoming\_text, incoming\_color, incoming\_bg):

# Text rendering in pygame works by rendering from a font.

# ( text, anti aliasing, color)

# If statement adds an option so we can add a backround to our text.

if incoming\_bg:

Text\_surface = ASSETS.FONT\_DEBUG\_MESSAGE.render(

incoming\_text, False,

incoming\_color, incoming\_bg)

else:

Text\_surface = ASSETS.FONT\_DEBUG\_MESSAGE.render(

incoming\_text, False,

incoming\_color)

return Text\_surface, Text\_surface.get\_rect()

def helper\_text\_height(font):

# Pass in a font, returns the height in pixels of the font.

# Allows to change fonts without hardcoding. ( text, anti aliasing, color)

font\_object = font.render('a', False, (0, 0, 0))

font\_rect = font\_object.get\_rect()

return font\_rect.height

#############################################

# Game

#############################################

def game\_main\_loop():

# In this function we loop the main game.

game\_quit = False

# player action definition

# Want to tell the game when player take action and when it can take an action

player\_action = "no-action"

while not game\_quit:

#handle player input

player\_action = game\_handle\_keys()

if player\_action == "QUIT":

game\_quit = True

if player\_action != "no-action":

for obj in GAME.current\_objects:

if obj.ai:

obj.ai.take\_turn()

# draw game

draw\_game()

CLOCK.tick(constants.GAME\_FPS)

pygame.quit()

exit()

def game\_handle\_keys():

# get player input

events\_list = pygame.event.get()

#process input

for event in events\_list: # loop through all events that have happened

if event.type == pygame.QUIT: # QUIT attribute - someone closed window

return "QUIT"

# Directions for arrow keys

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_UP:

PLAYER.creature.move(0, -1)

return "player-moved"

if event.key == pygame.K\_DOWN:

PLAYER.creature.move(0, 1)

return "player-moved"

if event.key == pygame.K\_LEFT:

PLAYER.creature.move(-1, 0)

return "player-moved"

if event.key == pygame.K\_RIGHT:

PLAYER.creature.move(1, 0)

return "player-moved"

# Swap maps ( ideal would be bound to a surface object)

if event.key == pygame.K\_h:

GAME.transition\_next()

return "no-action"

def game\_message(game\_msg, msg\_color):

GAME.message\_history.append((game\_msg, msg\_color))

def game\_initialize():

# This function initializes the main window, and pygame

global SURFACE\_MAIN, GAME, ASSETS, CLOCK, PLAYER, ENEMY, ENEMY2

# initialize pygame

pygame.init()

# set surface dimensions

SURFACE\_MAIN = pygame.display.set\_mode((

constants.MAP\_WIDTH\*constants.CELL\_WIDTH,

constants.MAP\_HEIGHT\*constants.CELL\_HEIGHT))

GAME = obj\_Game()

ASSETS = struc\_Assets()

CLOCK = pygame.time.Clock()

creature\_com1 = com\_Creature('Player')

PLAYER = obj\_Actor(1, 1, 'human' ,ASSETS.S\_PLAYER, creature = creature\_com1)

creature\_com2 = com\_Creature('Alien', death\_function = death\_monster)

ai\_com = ai\_Test()

ENEMY = obj\_Actor(24,24, 'alien', ASSETS.S\_ENEMY,

creature = creature\_com2, ai = ai\_com)

creature\_com3 = com\_Creature('Alien', death\_function = death\_monster)

ai\_com1 = ai\_Test()

ENEMY1 = obj\_Actor(12, 12, 'alien', ASSETS.S\_ENEMY,

creature = creature\_com3, ai = ai\_com1)

ai\_com2 = ai\_Test()

creature\_com4 = com\_Creature('Alien', death\_function = death\_monster)

ENEMY2 = obj\_Actor(5,5, 'alien', ASSETS.S\_ENEMY,

creature = creature\_com4, ai = ai\_com2)

creature\_com3 = com\_Creature('Alien Ghost', death\_function = death\_monster)

ai\_com3 = ai\_Test()

ENEMY3 = obj\_Actor(24,24, 'alien2', ASSETS.S\_ENEMY2,

creature = creature\_com3, ai = ai\_com3)

# Keeps track of time

CLOCK = pygame.time.Clock()

ASSETS = struc\_Assets()

GAME.current\_objects = [PLAYER, ENEMY, ENEMY1, ENEMY2]

if \_\_name\_\_ == '\_\_main\_\_':

game\_initialize()

game\_main\_loop()