SE 3XA3: Module Interface Specification T-Rex Acceleration

Team 15, Dev[enthusiasts]
Zihao Du (duz12)
Andrew Balmakund (balmakua)
Namit Chopra (choprn9)

April 9, 2021

Table 1: **Revision History**

Date	Version	Notes
03/18/2021	1.0	Initial Draft
04/07/2021	2.0	Revision 1

Model/Character

Uses

Pygame
Time

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
Character	pygame.display, pygame.image	Character	IllegalArgumentException
get_rect	—	pygame.rect	—
get_img	$\mathbb{Z}$	pygame.image	—
set_img	pygame.image $\mathbb{R}$, $\mathbb{R}$	—	IllegalArgumentException
set_ducking_img	—	pygame.image, $\mathbb{R}$, $\mathbb{R}$	IllegalArgumentException
duck	pygame.image ~~pygame.imge~~	—	IllegalArgumentException
get_ducking	—	$\mathbb{B}$	—
stand	pygame.image, ~~pygame.iamge~~	—	IllegalArgumentException
jump	pygame.image, ~~pygame.image~~	—	IllegalArgumentException
get_jumping	—	$\mathbb{B}$	—
checkbounds	—	—	—
invincible	~~pygame.image~~	—	~~IllegalArgumentException~~
get_invincible	—	$\mathbb{B}$	—
double_jump~~ing~~	—	—	—
get_double_jumping	—	$\mathbb{B}$	—
slo_mo	—	—	—
get_slo_mo	—	$\mathbb{B}$	—
is_powered	—	$\mathbb{B}$	—
update	pygame.image	—	— IllegalArgumentExcption
get_limit	—	$\mathbb{B}$	—
reset	pygam.image	—	—
get_power_time	—	$\mathbb{R}$	—
pause	—	—	—
resume	—	—	—

Semantics

State Variables

game_screen: pygame.display	//	Represents the screen
img: ~~pygame.image~~ list of pygame.image	//	Surface object with the image of the character
rect: pygame.rect	//	The rectangle of the character
is_ducking: $\mathbb{B}$	//	Is the character ducking
is_jumping: $\mathbb{B}$	//	Is the character jumping
is_invincible: $\mathbb{B}$	//	Is the character invincible
is_slo_mo: $\mathbb{B}$	//	Is the character slowing obstacles
is_double_jumping: $\mathbb{B}$	//	Can the character double jump
movement: $(\mathbb{R}, \mathbb{R})$	//	horizontal and vertical speed of the character
jumping_limit: $\mathbb{Z}$	//	The number of jumps
obtain_powerup_time: $\mathbb{R}$	//	The time when the character takes a powerup
puase_time: $\mathbb{R}$	//	Start time of a pause
pause_duration: $\mathbb{R}$	//	The duration of the pause

Environment Variables

None

State Invariant

$is\_ducking \wedge is\_jumping = False$
$is\_invincible \wedge is\_slo\_mo \wedge is\_double\_jumping = False$
$jumping\_limit < 3$

Assumptions

Constructor character is called first before other methods.

Access Routine Semantics

Character(screen, char_img):

- transition: game_screen, img, movement, jumping_limit. rect := screen, ~~char_img~~ char_img in NORMAL_SIZE, [0,0], 0, rectangle of char_image
  All Boolean state variables are initialized to False
  All time state variables are initialized to 0

- output: $out := self$
- exception:

  $exc := (img \equiv NULL) \Rightarrow IllegalArguementException$

  $exc := (game\_screen \equiv NULL) \Rightarrow IllegalArguementError$

get_rect():

- output: $out := rect$

get_image(img_number):

- output: $out := img[img\_number//IMAGE\_SELECTOR]$

set_img(new_img, bottom, left):

- transition: img, rect := ~~new_img~~ new_img in NORMAL_SIZE, rectangle of the new image with rect.bottom = bot; rect.left = left
- exception:

  $exc := (new\_img \equiv NULL) \Rightarrow IllegalArguementException$

set_ducking_img(new_img, bottom, left):

- transition: img, rect := new_img in DUCKING_SIZE, rectangle of the new image with rect.bottom = bot; rect.left = left
- exception:

  $exc := (new\_img \equiv NULL) \Rightarrow IllegalArguementException$

duck(ducking_img, ~~inv_ducking_img~~ char_img):

- transition: $\neg is\_jumping \Rightarrow (is\_ducking := False;$
  ~~$is\_invincible \Rightarrow img := inv\_ducking\_img \wedge \neg is\_invincible \Rightarrow img := ducking\_img)$~~
  $img, rect := char\_img$, rectangle of char_img)
- exception:

  $exc := (anyimg \equiv NULL) \Rightarrow IllegalArguementException$

get_ducking():

- output: $out := is\_ducking$

stand(~~inv_char~~, char_img):

- transition: $\neg is\_jumping \Rightarrow (is\_ducking := False;$
  ~~$is\_invincible \Rightarrow img := inv\_char \wedge \neg is\_invincible \Rightarrow img := char\_img$~~)
  $set\_img(char\_img, screen\_rect.left - Y\_OFFSET, screen\_rect.bottom - X\_OFFSET)$

- exception:

  $exc := (char\_img \equiv NULL) \Rightarrow IllegalArguementException$

jump(~~inv_jumping_char~~, jumping_img):

- transition: $\neg(is\_ducking \wedge is\_jumping) \Rightarrow (is\_jumping, movement[1], jumping\_limit :=$
  $True, JUMPING\_SPEED, jumping\_limit + 1 \wedge$
  $set\_img(char\_img, screen\_rect.left - Y\_OFFSET, screen\_rect.bottom - X\_OFFSET);$
  $is\_double\_jumping \wedge is\_jumping \wedge jumping\_limit \leq DOUBLE\_JUMPING\_LIMIT \Rightarrow$
  $jumping\_limit, movement[1] := jumping\_limit + 1, DOUBLEJUMPING\_SPEED$

- exception: $exc := (\_jumping\_img \equiv NULL) \Rightarrow IllegalArguementException$

get_jumping():

- output: $out := is\_jumping$

checkbounds():

- transition: $rect.bottom > screen\_rect.bottom - Y\_OFFSET \Rightarrow$
  $rect.bottom, is\_jumping, jumping\_limit := screen\_rect.bottom - Y\_OFFSET, False, 0$

invincible(~~inv_char~~):

- transition: $is\_invincible$,~~img~~,$is\_double\_jumping, is\_slo\_mo, obtain\_powerup\_time, pause\_duration$
  $:= True$,~~inv_char~~,$False, False, time(), 0$

- exception:

  $exc := (inv\_char \equiv NULL) \Rightarrow IllegalArguementException$

get_invincible():

- output: $out := is\_invincible$

double_jump():

- transition: $is\_invincible$,~~img,~~$is\_double\_jumping, is\_slo\_mo, obtain\_powerup\_time, pause\_duration := False$,~~inv_char,~~$True, False, time(), 0$

get_double_jump():

- output: $out := is\_double\_jumping$

slo_mo():

- transition: $is\_invincible$,~~img,~~$is\_double\_jumping, is\_slo\_mo, obtain\_powerup\_time, pause\_duration := False$,~~inv_char,~~$False, True, time(), 0$

get_slo_mo():

- output: $out := is\_slo\_mo$

is_powered():

- output: $out := is\_double\_jumping \vee is\_invincible \vee is\_slo\_mo$

update(char_img):

- transition: Change is_double_jumping or is_invincible or is_slo_mo to False and the corresponding image back if the power up has already lasted for DURATION_TIME. If is_jumping is True, decrease movement[1] by GRAVITY, and check if the jumping ends(if the character back to ground) If the character is not jumping or ducking, set the image to the original one
- exception: $exc := (char\_img \equiv NULL) \Rightarrow IllegalArguementException$

Local Constants

INIT_SPEED = ~~-20~~ $\mathbb{Z}$
GRAVITY = ~~1~~$\mathbb{Z}$
DURATION = ~~5~~$\mathbb{Z}$
NORMAL_SIZE = $(\mathbb{N}, \mathbb{N})$
DUCKING_SIZE = $(\mathbb{N}, \mathbb{N})$
X_OFFSET = $\mathbb{Z}$
Y_OFFSET = $\mathbb{Z}$
DOUBLEJUMPING_SPEED = $\mathbb{N}$
IMAGE_SELECTOR = $\mathbb{N}$
DOUBLE_JUMPING_LIMIT = $\mathbb{N}$
RESUME_TIME = $\mathbb{N}$

Model/Powerups

Uses

Pygame
Random
LoadAssets

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
Powerups	pygame.~~image~~display Powerups, $\mathbb{N}, \mathbb{N}, \mathbb{R}$	—	IllegalArguementException
get_rect	—	pygame.rect	—
get_width	—	$\mathbb{N}$	—
set_width	$\mathbb{N}$	—	IllegalArguementException
get_height	—	$\mathbb{N}$	—
set_height	$\mathbb{N}$	—	IllegalArguementException
get_speed	—	$\mathbb{R}$	—
set_speed	$\mathbb{N}$	—	—
get_img	—	~~string~~ $\mathbb{Z}$	—
set_img	pygame.img	—	IllegalArguementException
get_name	—	String	—
update	—	—	—

Semantics

State Variables

screen: pygame.display // Represents the game screen
name: ~~String~~ ~~$\mathbb{Z}$~~ // Represents the type of a power up
width: $\mathbb{N}$ // Represents the width of a power up
height: $\mathbb{N}$ // Represents the height of a power up
speed: $\mathbb{R}$ // Represents the speed of a power up
img: pygame.image // Surface object with a specified image drawn onto it
rect: pygame.rect // Represents the rectangle shape of the powerup
screen_rect: pygame.rect // Represents the game screen rectangle

State Invariant

~~None~~ $name \in [0, TYPES]$

Assumptions & Design Decisions

None

Access Routine Semantics

Powerups(~~powerup_img~~ window, width, height, speed):

- transition: width, height, speed, ~~img~~ screen, name, img := width, height, speed, ~~powerup_img,~~ window, random integer in [0,TYPES], corresponding image with given width and height
- output: $out := self$

get_rect():

- output: $out :=$ rect

get_width():

- output: $out :=$ width

set_width(new_width):

- transition: width := new_width
- exception: $exc :=$ (new_width < 0) $\Rightarrow IllegalArgumentException$

get_height():

- output: $out :=$ height

set_height(new_height):

- transition: height := new_height
- exception: $exc :=$ (new_height < 0) $\Rightarrow IllegalArgumentException$

get_speed():

- output:$out :=$ speed

set_speed(new_speed):

- transition:speed := new_speed

get_img():

- output: out := img

set_img(new_powerup_img):

- transition: img := new_powerup_img
- exception: $exc := (new\_powerup\_img \equiv NULL) \Rightarrow IllegalArgumentException$

get_name():

- output: out := name

update():

- transition: the object moves forward according to the speed

Local Constants

Y_OFFSET = $\mathbb{Z}$
TYPES = $\mathbb{Z}$

Model/Obstacle

Uses

Pygame

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
Obstacle	String,$\mathbb{N}$, $\mathbb{N}$, $\mathbb{R}$, pygame.image	Obstacle	—
get_width	—	$\mathbb{N}$	—
set_width	$\mathbb{N}$	—	IllegalArgumentException
get_height	—	$\mathbb{N}$	—
set_height	$\mathbb{N}$	—	IllegalArgumentException
get_speed	—	$\mathbb{R}$	—
set_speed	$\mathbb{R}$	—	IllegalArgumentException
get_img	—	pygame.image	—
set_img	pygame.image	—	IllegalArgumentException
get_rect	—	$(\mathbb{R}, \mathbb{R})$	
set_rect	$(\mathbb{R}, \mathbb{R})$	—	—
set_rect	$(\mathbb{R}, \mathbb{R})$	—	—

Semantics

State Variables

name: String	//	Represents the name of an obstacle
width: $\mathbb{N}$	//	Represents the width of an obstacle
height: $\mathbb{N}$	//	Represents the height of an obstacle
speed: $\mathbb{R}$	//	Represents the speed of an obstacle
img: pygame.image	//	Surface object with a specified image drawn onto it
rect: $(\mathbb{R}, \mathbb{R})$	//	X and Y coordinates of the obstacle

State Invariant

None

Assumptions & Design Decisions

None

Access Routine Semantics

Obstacle(name, width, height, speed, obstacle_img):

- transition: name, width, height, speed, img, rect:= name, width, height, speed, obstacle_img, obstacle_img.get_rect()
- output: $out := self$

get_width():

- output: $out :=$ width

set_width(new_width):

- transition: width := new_width
- exception: $exc :=$ (new_width < 0) $\Rightarrow IllegalArgumentException$

get_height():

- output: $out :=$ height

set_height(new_height):

- transition: height := new_height
- exception: $exc :=$ (new_height < 0) $\Rightarrow IllegalArgumentException$

get_speed():

- output: $out :=$ speed

set_speed(new_speed):

- transition: speed := new_speed
- exception: $exc :=$ (new_speed < 0) $\Rightarrow IllegalArgumentException$

get_img():

- output: $out :=$ img

set_img(new_osbtacle_img):

- transition: img := new_osbtacle_img
- exception: $exc := (new\_osbtacle\_img \equiv NULL) \Rightarrow IllegalArgumentException$

get_rect():

- output: *out* := rect

set_rect(x, y):

- transition: rect.x, rect.y := x, y

get_pos(x, y):

- output: out:= rect.x, rect.y

Model/DetectCollision

Uses

Pygame

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
detect_collision	pygame.sprite, seq of pygame.sprite	$\mathbb{B}$	—
~~find_collision~~ find_collision_obstacle	~~pygame.sprite, seq of pygame.sprite~~ Character, seq of Obstacle	Obstacle	—
find_collision_powerups	Character, seq of Powerups	Powerups	—

Semantics

State Variables

None

State Invariant

None

Assumptions & Design Decisions

All pygame.sprite objects have been defined have been defined.

Access Routine Semantics

detect_collision(character, elements):

- output: *out* := ($\forall$ element $\in$ elements | (character.X < element.X+element.width)$\wedge$ (character.X+character.width > element.X)$\wedge$(character.Y < element.Y+element.height)$\wedge$ (character.Y + Character.height > element.Y))

~~find_collision(character, elements):~~ find_collision_obstacle(character, obstacles):

- output: *out* := the ~~element~~ obstacle in the sequence ~~elements~~ obstacles that collides with the character.

find_collision_powerups(character, powerups):

- output: *out* := the powerup in the sequence powerups that collides with the character.

Model/UpdateEnvironment

Uses

Pygame
Random

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
update_floor	$\mathbb{Z}$, $\mathbb{Z}$	$\mathbb{Z}$	—
update_bg_colour	$\mathbb{Z}$, $\mathbb{Z}$, seq of $\mathbb{Z}$	seq of $\mathbb{Z}$	—

Semantics

State Variables

None

State Invariant

None

Assumptions

The game window and images are defined before any routines are called.

Access Routine Semantics

update_floor(floor_position, movement_speed):

- output: *out* := (floor_position $\leq$ BOUNDARY_LEFT) $\Rightarrow$ floor_position = 0 $\vee$ (floor_position = floor_position - ~~MOVEMENT_SPEED~~ movement_speed)

update_bg_colour(current_score, previous_score, bg_rgb):

- output: *out* := Every time the score is a multiple of CHANGE_BG_INTERVAL, the value of either red (bg_rbg[~~0~~ RED]), green (bg_rbg[~~1~~ GREEN]), or blue (bg_rbg[~~2~~ BLUE]) changes, which is selected randomly. The new value of one of the field is: ($0 \leq x \leq 2 |\ bg\_rgb[x] := (bg\_rgb[x] +$ CHANGE_BG_VAL $)\%$ MAX_RGB). The updated bg_rgb sequence is returned.

Local Constants

~~MOVEMENT_SPEED: 10~~
CHANGE_BG_VAL = ~~50~~ $\mathbb{N}$
MAX_RGB = $\mathbb{N}$
CHANGE_BG_INTERVAL = $\mathbb{N}$
RED = $\mathbb{N}$
GREEN = $\mathbb{N}$
BLUE = $\mathbb{N}$
BOUNDARY_LEFT = $\mathbb{Z}$

Model/Score

Uses

Time

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
Score	—	Score	—
get_current_score	—	$\mathbb{N}$	—
update_score	time.time	$\mathbb{N}$, $\mathbb{N}$	—
~~get_high_score~~ get_score	—	$\mathbb{N}$	—
reset_score	—	—	—
boost	—	—	—

Semantics

State Variables

high_score: $\mathbb{N}$
current_score: $\mathbb{N}$
previous_score: $\mathbb{N}$
~~start_time: $\mathbb{R}$~~
score_boost = $\mathbb{N}$

State Invariant

None

Assumptions

None

Access Routine Semantics

Score():

- output: *out* := *self*

- transition: high_score, current_score, ~~start_time~~ previous_score, boost := 0, 0, ~~current time~~ 0, 0

get_current_score():

- output: *out* := current_score

update_score(start_time):

- output: *out* := current_score, previous_score
- transition: previous_score, current_score, high_score := current_score, ((current time - start_time)· SCALE_FACTOR + score_boost · SCORE_BOOST_VAL) rounded to the nearest natural number, (current_score > high_score) ⇒ current_score ∀ ~~high_score~~

get_score():

- output: *out* := high_score

reset_score():

- transition: current_score, score_boost := 0, 0

boost():

- transition: score_boost := score_boost+1

Local Constants

SCALE_FACTOR = ~~5~~ ℕ
SCORE_BOOST_VAL = ℕ

Model/MainMenu

Uses

None

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
MainMenu	—	MainMenu	—
change_volume	$\mathbb{N}$, $\mathbb{N}$	—	—
get_volumes	—	$\mathbb{N}$, $\mathbb{N}$	—

Semantics

State Variables

background_music_volume: $\mathbb{N}$
sound_effects_volume: $\mathbb{N}$

State Invariant

None

Assumptions

None

Access Routine Semantics

MainMenu():

- output: *out* := self
- transition: background_music_volume, sound_effects_volume := MAX_VOLUME, MAX_VOLUME

change_volume(new_background_volume, new_sound_effects_volume):

- transition: background_music_volume, sound_effects_volume := new_background_volume, new_sound_effects_volume

~~get_volumes():~~

- ~~output: *out* := background_music_volume, sound_effects_volume~~

~~Local Constants~~

~~MAX_VOLUME: 100~~

View/DisplayObstacle

Uses

Pygame
Time
Obstacle
Random
DetectCollision

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
DisplayObstacle	pygame.display	DisplayObstacle	IllegalArgumentException
get_obstacle_list	—	seq of Obstacle	—
remove_obstacle	Obstacle	—	—
generate_obstacle	$\mathbb{R}, \mathbb{R}$ seq of Obstacle, $\mathbb{R}$, seq of Powerups	$\mathbb{R}$	—
draw_obstacle	$\mathbb{R}, \mathbb{R}$, Obstacle	—	—
update_obstacle_display	—	—	—
update_speed	$\mathbb{R}$	—	IllegalArgumentException

Semantics

State Variables

game_screen: pygame.display
obtacle_list: seq of Obstacle

State Invariant

None

Assumptions

None

Access Routine Semantics

DisplayObstacle(window):

- output: *out* := *self*

- transition: game_screen, obstacle_list := window, []
- exception: *exc*: (game_screen = NULL ⇒ IllegalArgumentExcpetion)

get_obstacle_list():

- output: *out* := obstacle_list

remove_obstacle(obstacle):

- transition: obstacle_list := obstacle_list - obstacle.

draw_obstacle(current_x, current_y, obstacle):

- transition: game_screen := Draw the given obstacle at a specific location (X, Y) coordinates using the ~~obstacle image~~ obstacle.get_img().

generate_obstacle(current_x, current_y, type_of_obstacles, prev_obstacle_spawn_time, generated_powerups):

- transition: Generate a random kind of obstacle after APPROPRIATE_TIME ~~and~~ , add it to the obstacle_list and draw it onto the screen. Also checks for any instance of a collision with an already generated powerups so that an obstacle and poweup don't overlap.
- out: *out* := current time

update_obstacle_display():

- transition: For each obstacle in obstacle_list, draw each obstacle at its new position by considering the speed of each object. Once the obstacle is outside the boundaries of the game_screen, remove the obstacle from the list.

update_speed(new_speed):

- transition: For each obstacle in obstacle_list, change the speed to new_speed.

Local Constants

APPROPRIATE_TIME = $\mathbb{Z}$

View/DisplayPowerups

Uses

Pygame
Powerups
Random
Time
DetectCollision
Obstacle

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
DisplayPowerups	pygame.display	DisplayPowerups	IllegalArgumentException
get_powerups_list	—	—	—
remove_powerups_list	Powerups	—	—
generate_powerup	$\mathbb{R}$, seq of Obstacles, $\mathbb{R}$	—	—
draw_powerups	Powerups	—	—
update_powerups	seq of Obstacle	—	—
update_speed	$\mathbb{R}$	—	—

Semantics

State Variables

powerups_diplayed: seq of Powerups
game_screen: pygame.display
generate_time: $\mathbb{R}$

State Invariant

None

Assumptions

None

Access Routine Semantics

display_powerup(game_screen):

- output: *out* := *self*
- transition: game_screen, powerups_diplayed, generate_time := game_screen, $\coprod$, pygame.sprite.Group(), time()
- exception: *exc*: (game_screen = NULL $\Rightarrow$ IllegalArgumentExcpetion)

get_powerups_list():

- out: *out* := powerups_displayed

remove_powerups_list(p):

- transition: powerups_displayed := powerups_displayed.remove(p)

generate_powerup(speed, obstacles, obstacle_spawn_time):

- transition: powerups_diplayed := Add a random kind of powerup in the powerups_displayed (a list of powerups displayed on the game_screen) , update generate_time and draw it if there is no overlapping between obstacles and the new object and $time() - generate\_time > randint(RAND\_MIN, RAND\_MAX)$ $and time() - obstacle\_spawn\_time \geq INTERVAL\_TIME$.

draw_powerups(p):

- transition: draw Powerup p on the screen

update_powerups(obstacles):

- transition: all elements in powerups_displayed are updated by position and drawn. If there is overlapping, remove the powerup, and if the powerup goes out of the screen , get rid of it.

update_speed(speed):

- transition: $\forall p \in powerups\_displayed : p.set\_speed(speed)$

Local Constants

POWERUPS_WIDTH = $\mathbb{N}$
POWERUPS_HEIGHT = $\mathbb{N}$
INTERVAL_TIME = $\mathbb{N}$
RANDOMNESS = $\mathbb{R}$
RAND_MIN = $\mathbb{N}$
RAND_MAX = $\mathbb{N}$

View/DisplayEnvironment

Uses

~~UpdateEnvironment~~
Pygame

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
DisplayEnvironment	pygame.display	DisplayEnvironment	IllegalArgumentException
draw_score	Score	—	—
~~display_instruction~~ display_msg	String, seq of $\mathbb{Z}$	—	—
draw_floor	pygame.image, $\mathbb{Z}$	—	IllegalArgumentException
draw_background	pygame.image, seq of $\mathbb{Z}$	—	IllegalArgumentException
display_powerup	~~pygame.image~~ time.time	—	IllegalArgumentException

Semantics

State Variables

None

State Invariant

game_screen: pygame.display

Assumptions

None

Access Routine Semantics

DisplayEnvironment(window):

- output: $out := self$
- transition: game_screen := window
- exception: $exc := (window \equiv NULL) \Rightarrow IllegalArgumentException$

draw_score(score):

- transition: The score is drawn on the game_screen using the display_msg method.

~~draw_instruction (instructions)~~ display_msg(msg, msg_pos):

- transition: ~~The instruction is shown on the screen for TIME and disappears after~~ Draws msg on the game_screen at the position msg_pos.

draw_floor(floor, floor_position):

- transition : Draw the floor onto the game_screen at floor_position.
- exception: $exc := (floor \equiv NULL) \Rightarrow IllegalArgumentException$

draw_background(background, bg_rgb):

- transition : Fills the game_screen with the color bg_rgb and draws ~~Draw the~~ background ~~img~~ onto the game_screen.
- exception: $exc := (background \equiv NULL) \Rightarrow IllegalArgumentException$

draw_powerup(time):

- transition : Draws the time (remaining time left on acquired powerup) on the game_screen.

~~Local Constants~~

~~TIME = 5 seconds~~

View/DisplayWindow

Uses

Pygame

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
DisplayWindow	—	DisplayWindow	—
get_game_screen	—	pygame.display	—

Semantics

State Variables

game_screen: pygame.display

State Invariant

None

Assumptions

None

Access Routine Semantics

DisplayWindow():

- output: $out := self$
- transition: game_screen := a new pygame window with dimensions WIDTH $\times$ HEIGHT pixels

get_game_screen():

- output: out := game_screen

Local Constants

WIDTH = ~~800~~ $\mathbb{N}$
HEIGHT = ~~600~~ $\mathbb{N}$

View/DisplayCharacter

Uses

Character

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
DisplayCharacter	pygame.display, Character	—	—
draw_character	—	—	—

Semantics

State Variables

game_screen: pygame.display
game_character: Character
step: $\mathbb{Z}$

State Invariant

~~None~~ $0 \leq step \leq FRAME$

Assumptions

None

Access Routine Semantics

DisplayCharacter(window, character):

- transition: game_screen, game_character, step := window, character, 0

draw_character():

- transition: draw the character onto the game_screen according to step to mimic a gif.
  $step \leq FRAME \Rightarrow step := 0; step := step + 1$

Local Constants

FRAME = $\mathbb{N}$

View/PlaySound

Uses

~~LoadAssets~~
Pygame

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
PlaySound	seq of pygame.mixer.Sound	PlaySound	—
get_sound_effect	—	$\mathbb{R}$	—
set_sound_effect	$\mathbb{R}$	—	IllegalArgumentException
get_background	—	$\mathbb{R}$	—
set_background	$\mathbb{R}$	—	IllegalArgumentException
play_bg_music	—	—	—
play_jump_sound	—	—	—
play_duck_sound	—	—	—
play_collision_sound	—	—	—
play_powerup_sound	—	—	—
play_game_over_sound	—	—	—
stop_music	—	—	—

Semantics

State Variables

background_music: pygame.mixer.Sound
jump_sound: pygame.mixer.Sound
duck_sound: pygame.mixer.Sound
collision_sound: pygame.mixer.Sound
powerup_pickup_sound: pygame.mixer.Sound
game_over_sound: pygame.mixer.Sound
sound_effect_vol: $\mathbb{R}$
background_vol: $\mathbb{R}$

State Invariant

None

Assumptions

None

Access Routine Semantics

PlaySound(sound_list):

- output: $out := self$
- transition: background_music, jump_sound, duck_sound, collision_sound, powerup_pickup_sound, game_over_sound, sound_effect_vol, background_VOL := the assets are loaded in from the LoadAssets module (for background, jump, duck, collision, powerup pickup and game over), SOUND_EFFECT_VOL_INTIAL, BACKGROUND_VOL_INTIAL.

get_sound_effect():

- output: $out :=$ sound_effect_vol

set_sound_effect(new_vol):

- transition: sound_effect_vol := new_vol
- exception: $exc := (0 < new\_vol \lor new\_vol > 1) \Rightarrow IllegalArgumentException$

get_background():

- output: $out :=$ background_vol

set_background(new_vol):

- transition: background_vol := new_vol
- exception: $exc := (0 < new\_vol \lor new\_vol > 1) \Rightarrow IllegalArgumentException$

play_bg_music():

- transition: plays the background music at background_vol.

play_jump_sound():

- transition: plays the jump sound effect at sound_effect_vol.

play_duck_sound():

- transition: plays the duck sound effect at sound_effect_vol.

play_collision_sound():

- transition: plays the collision sound effect at sound_effect_vol.

play_powerup_sound():

- transition: plays the powerup pickup sound effect at sound_effect_vol.

play_game_over_sound():

- transition: plays the game over sound effect at sound_effect_vol.

stop_music():

- transition: stops all current audio that is playing.

Local Constants

SOUND_EFFECT_VOL_INITIAL = $\mathbb{R}$
BACKGROUND_VOL_INITIAL= $\mathbb{R}$

View/DisplayMenu

Uses

~~MainMenu~~
Pygame

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
DisplayMenu	pygame.display	DisplayMenu	IllegalArgumentException
display_main_menu	pygame.image	—	—
display_pause_menu	pygame.image	—	—
~~display_exit_menu~~ display_end_menu	pygame.image	—	—
display_setting_menu	pygame.image	—	—
display_resume_menu	$\mathbb{N}$	—	—
display_instruction_menu	pygame.image	—	—

Semantics

State Variables

game_screen: pygame.display

State Invariant

None

Assumptions

None

Access Routine Semantics

DisplayMenu(window):

- output: $out := self$
- transition: game_screen := window
- exception: $exc := (window \equiv NULL) \Rightarrow IllegalArgumentException$

display_main_menu(main_menu_img):

- transition: game_screen := ~~Main menu~~ Display the main menu image which contains layout for ~~with~~ 'Play', 'Quit' and 'Setting' ~~buttons~~. The image will also display the keybindings that correspond to accessing each menu.

display_pause_menu(pause_menu_img):

- transition: game_screen := ~~Pause menu~~ Display the pause menu image which contains ~~with~~ 'Resume' ~~button~~ to resume back to the current game or 'Exit' ~~button~~ to go to the main menu. The image will also display the keybindings that correspond to accessing each menu.

~~display_exit_menu~~ display_end_menu(exit_menu_img):

- transition: game_screen := ~~Exit menu~~ Display the exit menu image which contains ~~with~~ 'Return' ~~button~~ to the main menu after game session has ended and 'Quit' to quit the game application (terminating the game). The image will also display the keybindings that correspond to accessing each menu.

display_setting_menu(setting_menu_img):

- transition: game_screen := ~~Setting menu that can be used to change the volume and theme.~~ Display the setting menu image and also contains buttons which are plus and minus to change the background volume and sound effect volume. The img will also contain a 'Back' to return to the main menu and 'Confirm' to save the current changes of background and sound effect volume setting

display_resume_menu(time_remaining):

- transition: game_screen := Displays the current time_ remaining on the screen.

display_instruction_menu(instruction_menu_img):

- transition: game_screen := Displays the instruction menu image on the screen which contains information about the game and how to play it. It also contains information about the keybinding to return back to the main menu.

View/LoadAssets

Uses

Pygame

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
load_floor	—	pygame.image	IllegalArugementException
load_background	—	pygame.image	IllegalArugementException
load_character	—	seq of pygame.image	IllegalArugementException
load_all_obstacle	—	seq of pygame.image	IllegalArugementException
load_all_powerups	—	seq of pygame.image	IllegalArugementException
load_main_menu	—	pygame.image	IllegalArugementException
load_pause_menu	—	pygame.image	IllegalArugementException
load_end_menu	—	pygame.image	IllegalArugementException
load_setting_menu	—	pygame.image	IllegalArugementException
load_instruction_menu	—	pygame.image	IllegalArugementException
load_sound	—	pygame.mixer	IllegalArugementException

Semantics

State Variables

None

State Invariant

None

Assumptions

All the files are in the appropriate directory with proper names and format.

Access Routine Semantics

load_floor():

- output: *out* := returns a Pygame image object with the floor image loaded in.

- exception: $exc := (\text{FLOOR\_IMG} \equiv \neg FileExists) \Rightarrow FileNotFoundError$

load_background():

- output: $out :=$ returns a Pygame image object with the background image loaded in.
- exception: $exc := (\text{BACKGROUND\_IMG} \equiv \neg FileExists) \Rightarrow FileNotFoundError$

load_character():

- output: $out :=$ returns a sequence of Pygame image objects with images of ~~different characters~~ the character performing different actions (including jumping, ducking, and running) loaded in.
- exception: $exc := (\text{CHARACTER\_IMG} \equiv \neg FileExists) \Rightarrow FileNotFoundError$

load_all_obstacles():

- output: $out :=$ returns a sequence of Pygame image objects with all obstacle images loaded in.
- exception: $exc := (\text{OBSTACLE\_IMG} \equiv \neg FileExists) \Rightarrow FileNotFoundError$

load_all_powerups():

- output: $out :=$ returns a sequence of Pygame image objects with the all powerup images loaded in.
- exception: $exc := (\text{POWERUP\_IMG} \equiv \neg FileExists) \Rightarrow FileNotFoundError$

load_main_menu():

- output: $out :=$ returns a Pygame image object with the main menu image loaded in.
- exception: $exc := (\text{MAINMENU\_IMG} \equiv \neg FileExists) \Rightarrow FileNotFoundError$

load_pause_menu():

- output: $out :=$ returns a Pygame image object with the pause menu image loaded in.
- exception: $exc := (\text{MAINMENU\_IMG} \equiv \neg FileExists) \Rightarrow FileNotFoundError$

load_end_menu():

- output: *out* := returns a Pygame image object with the end menu image loaded in.
- exception: *exc* := (MAINMENU_IMG ≡ ¬*FileExists*) ⇒ *FileNotFoundError*

load_setting_menu():

- output: *out* := returns a Pygame image object with the setting menu image loaded in.
- exception: *exc* := (MAINMENU_IMG ≡ ¬*FileExists*) ⇒ *FileNotFoundError*

load_instruction_menu():

- output: *out* := returns a Pygame image object with the instruction menu image loaded in.
- exception: *exc* := (MAINMENU_IMG ≡ ¬*FileExists*) ⇒ *FileNotFoundError*

load_sound():

- output: *out* := returns Pygame sound objects with the background music and sound effects loaded in.
- exception: *exc* := (SOUND_MP3 ≡ ¬*FileExists*) ⇒ *FileNotFoundError*

Local Constants

FLOOR_IMG = ~~'floor.png'~~ image
BACKGROUND_IMG = ~~'background.png'~~ image
CHARACTER_IMG = ~~['character.png', 'character_invisible.png', 'character_slomo.png']~~ seq of image
OBSTACLE_IMG = ~~['obstacle1.png', 'obstacle2.png']~~ seq of image
POWERUP_IMG = ~~['powerup1.png', 'powerup2.png', 'powerup3.png', 'powerup4.png']~~ seq of image
MAINMENU_IMG = ~~'mainmenu.png'~~ image
SOUND_MP3 = ~~['sound1.mp3', 'sound2.mp3', 'sound3.mp3', 'sound4.mp3', 'sound5.mp3']~~ seq of audio

Controller/GameController

Uses

Pygame
Time
sys
MenuController
Character
Obstacle
Powerups
DetectCollision
UpdateEnvironment
Score
DisplayObstacle
DisplayPowerups
DisplayEnvironment
DisplayWindow
DisplayCharacter
PlaySound
DisplayMenu
LoadAssets

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
GameController	—	GameController	—
check_user_input	—	String	—
game_loop	clock(), $\mathbb{R}$	—	—
increase_game_speed	DisplayPowerups, DisplayObstacles	—	—
main_menu	pygame.font	String	—
run_game	—	—	—
resume_game	DisplayObstacles, DisplayEnvironment, DisplayPowerups, DisplayCharacter, seq of $\mathbb{Z}$, $\mathbb{R}$	$\mathbb{R}, \mathbb{R}$	—
detect_powerups_collision	DisplauPowerups, DisplayObatcles	—	—
detect_obstacles_collision	$\mathbb{B}, \mathbb{R}$, DisplayObatcles	$\mathbb{B}$	—

Semantics

State Variables

game_screen: pygame.display
obstacle_img: seq of pygame.image
sound_list: seq of pygame.mixer
game_speed: $\mathbb{R}$
load_character: seq of seq of pygame.image
obstacle_list: seq of Obstacle
character : Character
powerup_list : seq of Powerups
play_sound: PlaySound
menu_controller: MenuController
pause_time: $\mathbb{R}$
~~background_music: pygame.mixer~~
~~sound_effects: pygame.mixer~~
main_menu_img: pygame.image
pause_menu_img: pygame.image
end_menu_img: pygame.image
setting_menu_img: pygame.image

instruction_menu_img: pygame.image
floor: pygame.image
floor_position: $\mathbb{N}$
background: pygame.image
score_count: Score
is_paused: $\mathbb{B}$

Environment Variables

QUIT: Mouse Input close the window
K_DOWN: Keyboard input down arrow
K_UP: Keyboard input up arrow
K_p: Keyboard input p button
K_s: Keyboard input s button
K_h: Keyboard input h button
K_SPACE: Keyboard input space bar

State Invariant

~~None~~$game\_speed \in [INIT\_SPPED, MAX\_SPEED]$

Assumptions

~~None~~ The constructor shall be called before other methods

Access Routine Semantics

GameController():

- transition: game_screen := DisplayWindow.get_game_screen()
  obstacle_list, ~~powerup_list~~ := seq of all kinds of Obstacles, []
  game_speed := INIT_SPEED
  character := Character(game_screen, LoadAssets.load_character()[0])
  ~~background, sound_effects := LoadAssets.load_sound()~~
  ~~floor := LoadAssets.load_floor()~~
  ~~background := LoadAsset.load_background()~~
  All images, sounds are initialized with corresponding LoadAssets methods
  menu_controller := MenuController(game_screen)
  pause_time := 0

floor_position := 0
is_puased := False
score_count := Score()
play_sound.play_bg_music()

- output: *out* := *self*

check_user_input():

- transition: Checks for user input and calls the corresponding method. The method is responsible for handling input for character control and moving the character based on the input. The method also handles the user inputs for starting, quitting, and going to the settings from the main menu.
- output: *out* := *User input in pause menu*

Input Key	Behaviour
pygame.QUIT	System exit
pygame.KEYDOWN	If any keyboard input is pressed
pygame.K_DOWN pygame.K_UP ~~pygame.K_Up when jumping~~ pygame.K_p	character.duck() & play_duck_sound() character.jump() & play_jump_sound() ~~character.double_jump() & play_jump_sound()~~ MenuController.pause_menu(game_screen)
pygame.KEYUP	If any keyboard input is not pressed
pygame.K_DOWN	character.stand()

game_loop(clock, game_start_time):

- transitions: It is the main game loop of the game that will continuously run until the current game session ends when the user presses the 'Quit' button in the main menu. It will call methods from other modules to control the game play. ~~The floor image will have its position constantly moving and being updated. The events of each keyboard input will be constantly monitored to ensure the user input is registered. The score will be incremented and depending on the current score, the background color will change. There will be random types of obstacles and powerups generating at random times. There will be a constant check for collision detection between the character and obstacle, and character and powerups. If a collision between a character and obstacle has occured, the current game state stops and goes to the exits menu to compare current score and highest score achieved.~~ It calls check_user_input method to detect input, and gets all objects drawn on the screen and updated every loop. It also detects collision with detectDollision methods and takes corresponding action if collision happens.

increase_game_speed(powerups, obstacles)

- transition: $game\_speed < MAX\_SPEED \Rightarrow (character.get\_slo\_mo \Rightarrow gamespeed := INIT\_SPEED \wedge$
  $\neg character.get\_slo\_mo \Rightarrow gamespeed := INIT\_SPEED + SPEED\_FACTOR * (score//SCORE\_SPEED))$
  $obsacles.update\_speed(game\_speed); powerups.update\_speed(game\_speed)$

main_menu(font)

- transition: It detects all user input in main menu and corresponds to these inputs.
- out: *out* := *action*

Input Key	Behaviour
pygame.KEYDOWN	If any keyboard input is pressed
pygame.K_SPACE	action = "Play"
pygame.K_q	stop_music & running = False
pygame.K_s	call menu_controller setting menu method
pygame.K_h	call menu_controller instruction menu method

run_game()

- transition: start the game if action is "Play", initialize font, clock and Boolean value running.

resume_game(d_obstacles, d_environment, d_powerups, d_character, bg_rgb, game_start_time)

- transition: Resume and redrawn all elments in game, show a RESUME_TIME count down before the game is resumed.
- output: *out* := *game_start_time, obstacle_spawn_time*

detect_powerups_collision(d_powerups, d_obstacles)

- transition: Detect collision between the character and powerups. If there is a collision, play sound, change the status of the character/score) and remove that powerup.

detect_obstacles_collision(running, current_score, d_obstacles)

- transition: Detect the collision between obstacles and character and play sound for collisions. If there is a successful collision(with no invincibility powerup),set running to False
- out: *out* := *running*

Local Constants

$\text{INIT\_SPEED} = \mathbb{N}$
$\text{MAX\_SPEED} = \mathbb{N}$
$\text{OBS\_START\_X} = \mathbb{N}$
$\text{OBS\_START\_Y} = \mathbb{N}$
$\text{INIT\_BG} = [\mathbb{N}, \mathbb{N}, \mathbb{N}]$
$\text{FONTSIZE} = \mathbb{N}$
$\text{FPS} = \mathbb{N}$
$\text{POWERUPS\_TIME} = \mathbb{N}$
$\text{RESUME\_TIME} = \mathbb{N}$
$\text{SPEED\_FACTOR} = \mathbb{R}$
$\text{SCORE\_SPEED} = \mathbb{N}$

Controller/MenuController

Uses

~~MainMenu~~ DisplayMenu
Pygame

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
MenuController	pygame.display	MenuController	IllegalArgumentException
setting_menu	~~pygame.display~~ PlaySound, pygame.image	—	—
pause_menu	~~pygame.display~~ pygame.image	—	—
~~exit_menu~~ end_menu	~~pygame.display~~ $\mathbb{N}$, $\mathbb{N}$, pygame.image	—	—
resume_menu	$\mathbb{N}$	—	—
instruction_menu	pygame.image	—	—

Semantics

State Variables

game_screen: pygame.display

Environment Variables

QUIT: Mouse Input close the window
K_b: Keyboard input b arrow
K_r: Keyboard input r button
K_c: Keyboard input c button
K_q: Keyboard input q button

State Invariant

None

Assumptions

None

Access Routine Semantics

MenuController(window):

- output: $out := self$
- transition: game_screen := window
- exception: $exc := (window \equiv NULL) \Rightarrow IllegalArgumentException$

setting_menu(~~screen~~ play_sound, setting_menu_img):

- transition: Display the settings menu onto the screen and handle the volume ~~and theme~~ change according to user input. The user will be able to modify the sound effect and background volume separately and handle saving these options. When the user press the 'Back' button, the method terminates and control is shifted back to the game controller.

pause_menu(~~screen~~, pause_menu_img):

- transition: Display the pause menu onto the screen and freeze the current game state. The score count is kept unchanged and all the elements of the game (obstacles, character, and background) stop moving. If the 'Resume' button is pressed, the game data current state is unfrozen, the method terminates and the control is shifted back to the game controller. If the Exit' button is pressed, the method terminates and the control is shifted back to the game controller (without saving the current game state upon exiting).

~~exit_menu(screen)~~ end_menu(current_score, highest_score, end_menu_img):

- transition: Display the exit menu onto the screen and restart or quit the game according to user input. As well display the current_score and highest_score onto the screen. When the user presses the 'Exit' button, the game ~~goes back to main menu~~ terminates. If the 'Restart' button is pressed, ~~a new game starts~~ the method terminates and control is passed back to the game controller to return back to main menu ~~whatever button is pressed~~.

resume_menu(time_remaining):

- transition: Display the time remaining before the game resumes back to the current game state where all character, powerups, obstacles and other environment visuals are still visible.

instruction_menu(instruction_menu_img):

- transition: Display the instruction menu onto the screen and handle the input to return back to the main menu. If the 'Back' button is pressed, the method terminates and the control is shifted back to the game controller.