SE 3XA3: Module Interface Specification
Super Refactored Mario Python

203, Abstract Connoisseurs
David Jandric, jandricd
Daniel Noorduyn, noorduyd
Alexander Samaha, samahaa

March 14, 2020

Entity Base Module

Uses

Vector2D
pygame.Rect // Class for representing a rectangle

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
new entity_base	$\mathbb{Z}, \mathbb{Z}, \mathbb{R}$	entity_base	—
apply_gravity	—	—	—
update_traits	—	—	—
get_pos_index	—	Vector2D	—
get_float_pos_index	—	Vector2D	—

Semantics

State Variables

vel: Vector2D // Represents velocity of the entity
rect: Rect // Represents the rectangle the entity is encased in
gravity: $\mathbb{R}$ // Represents the gravitational acceleration of the entity
traits: List[Trait] // List of traits the entity has
alive: $\mathbb{B}$ // Self explanatory
time_after_death: $\mathbb{R}$ // Represents the time after an entity has died
timer: $\mathbb{N}$ // Keeps track of the number of time the entity has been updated
type: string // Represents the name of the type of entity
on_ground: $\mathbb{B}$ // Self explanatory
obey_gravity: $\mathbb{B}$ // Self explanatory

State Invariant

None

Assumptions & Design Decisions

None

Access Routine Semantics

new entity_base(x, y, gravity):

- transition:

  vel, rect, gravity := Vector2D(0, 0), Rect(x * 32, y * 32, 32, 32)
  traits, alive, on_ground, obey_gravity := None, True, False, True
  timer_after_death, timer, type := 5, 0, ””

- output: *out* := *self*

apply_gravity():

- transition:

obey_gravity	¬ on_ground ⇒ vel := vel + Vector2D(0, gravity)
	on_ground ⇒ vel.set_y(0)

update_traits():

- transition: If there are traits, then update all traits using trait.update()

get_pos_index():

- output: *out* := Vector2D(int(rect.x / 32), int(rect.y / 32))

get_float_pos_index():

- output: *out* := Vector2D(rect.x / 32, rect.y / 32)

Goomba Module

Uses

Animation
Camera
EntityBase
Level
pygame.Surface

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
new goomba	Surface, Map[string: Surface | Animation], $\mathbb{R}, \mathbb{R}$, Level	entity_base	—
update	Camera	—	—
draw_goomba	Camera	—	—
on_dead	Camera	—	—
draw_flat_goomba	Camera	—	—

Semantics

State Variables

sprite_collection: Map[string: Surface — Animation]	//	Collection of all sprites
animation: Animation	//	Represents the images related to Koopa animation
screen: Surface	//	Represents the entire screen
type: string	//	The type of the entity
dashboard: Dashboard	//	Represents the dashboard

State Invariant

None

Assumptions & Design Decisions

None

Access Routine Semantics

new goomba(screen, sprite_coll, x, y, level):

- transition:

  sprite_collection := sprite_coll

  animation := A new animation object, initialized with the images related to the Goomba

  screen, type, dashboard := screen, "Mob", level.dashboard

- output: $out := self$

update(camera):

- transition: If the Goomba is alive, then apply gravity (using apply_gravity()) and draw the Goomba (using draw_goomba(camera)). If the Goomba is dead, then call on_dead(camera).

draw_goomba(camera):

- transition: screen.blit(animation.image, (rect.x + camera.x, rect.y)), animation.update()

on_dead(camera):

- transition: When killed, the Goomba will draw a string representing the number of points given by killing the Goomba, and also replace the regular animation images of the Goomba with the flat image. Then, after one cycle of this, it will set the alive attribute to None, deleting the Goomba.

draw_flat_goomba(camera):

- transition: Draws the flat Goomba to the screen.

Koopa Module

Uses

Animation
Camera
EntityBase
Level
pygame.Surface

Syntax

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new koopa	Surface, Map[string: Surface | Animation], $\mathbb{R}, \mathbb{R}$, Level	entity_base	—
update	Camera	—	—
draw_koopa	Camera	—	—
shell_bouncing	Camera	—	—
die	Camera	—	—
sleeping_in_shell	Camera	—	—
update_alive	Camera	—	—

Semantics

State Variables

sprite_collection: Map[string: Surface — Animation]	//	Collection of all sprites
animation: Animation	//	Represents the images related to Koopa animation
screen: Surface	//	Represents the entire screen
type: string	//	The type of the entity
dashboard: Dashboard	//	Represents the dashboard

State Invariant

None

Assumptions & Design Decisions

None

Access Routine Semantics

new koopa(screen, sprite_coll, x, y, level):

- transition:

  sprite_collection := sprite_coll

  animation := A new animation object, initialized with the images related to the Koopa

  screen, type, dashboard := screen, ”Mob”, level.dashboard

- output: $out := self$

update(camera):

- transition: If the Koopa is alive, then call update_alive(camera). If the Koopa is sleeping, then call update_sleeping(camera). If the Koopa is in it’s shell bouncing state, call shell_bouncing(camera). If the Koopa is dead, then call die(camera)

draw_koopa(camera):

- transition: Draw the Koopa on the screen, using previously mentioned methods.

shell_bouncing(camera):

- transition: When the Koopa is in this state, it will bounce back and forth, and obey gravity. The animation image of the Koopa is set to the hiding image, then draw_koopa(camera) is called.

die(camera):

- transition: When Koopa is killed, display the points on the screen, and draw the hiding Koopa. After 500 frames, the Koopa is deleted by setting alive := None

sleeping_in_shell(camera):

- transition: If the timer time_after_death, then draw the Koopa hiding image. Otherwise, set alive, timer := True, 0. Then, increment timer.

update_alive(camera):

- transition: Call apply_gravity, draw_koopa(camera), animation.update()

Mario Module

Module

Uses

Uses entity_base

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Mario	$\mathbb{N}, \mathbb{N}$, Level, Screen, Dashboard, $\mathbb{R}$	Mario	
get_pos		$\mathbb{N}, \mathbb{N}$	
set_pos	$\mathbb{N}, \mathbb{N}$		TypeError
update			
move_mario			
check_entity_collision			
on_collision_with_item	Item		TypeError
on_collision_with_block	random_block		TypeError
on_collision_with_mob	entity_base, collision_state		TypeError
bounce			
kill_entity	entity_base		TypeError
death_in_game			
game_over			
get_lives		$\mathbb{N}$	

Semantics

State Variables

sprite_collection: Object of type Sprites
camera: Object of type Camera
input: Object of type Input
in_air: $\mathbb{B}$
in_jump: $\mathbb{B}$

animation: Object of type Animation
traits: Seq of Traits
level_obj: Object of type Level
collision: Object of type Collider
screen: Object of type Display
entity_collider: Object of type EntityCollider
dashboard: Object of type Dashboard
restart: $\mathbb{B}$
pause: $\mathbb{B}$
pause_obj: Object of type Pause
lives: $\mathbb{N}$

State Invariant

None

Assumptions & Design Decisions

- The Mario constructor is called before any other access routines are called. Once called, the constructor will then not be used again.

Access Routine Semantics

new Mario(x, y, level, screen, dashboard, gravity):

- transition: sprite_collection = Sprites().sprite_collection
  camera = new Camera(rect, self)
  input = new Input(self)
  in_air = False
  in_jump = False
  animation = new Animation(Seq of sprite_collection)
  traits =
  level_obj = level
  collision = Collider(self)
  screen = screen
  EntityCollider = EntityCollider(self)
  dashboard = dashboard
  restart = False
  pause = False

pause_obj = Pause(screen, self, dashboard)
lives = ℕ

update():

- transition: updates the following functions update_traits, move_mario, camera, gravity, check_entity_collision and check_for_input.
- exception: None

move_mario():

- transition: x, y := x + vel.get_x, y + vel.get_y
- exception: None

check_entity_on_collision():

- transition: Checks if Mario collided with either of Item, Block or Mob entity and redirects to appropriate function.
- exception: None

on_collision_with_item(item):

- transition: Collided item is removed from list of current items, dashboard.points increased by 100, dashboard.coins increased by 1.
- exception: TypeError if item is not of type Item

on_collision_with_block(block):

- transition: Collided item is removed from list of current items, dashboard.points increased by 100, dashboard.coins increased by 1.
- exception: TypeError if block is not of type RandomBlock

on_collision_with_mob(mob, collision_state):

- transition: Collided block is activated, dashboard.coins is incremented by 1.
- exception: TypeError if mob is not of type entity_base

bounce():

- transition: traits["bounceTrait"].jump := True
- exception: None

kill_entity(ent):

- transition: If the entity is not a Koopa, then ent.alive := False, otherwise ent.alive := "sleeping". Dashboard.points is further incremented 100 points.
- TypeError if ent is not of type entity_base

game_over():

- transition: The screen is filled with black excluding a small circle around the player character. self.restart := True.
- exception: None

get_pos():

- output: camera.x + rect.x, y
- exception: None

set_pos(x, y):

- transition: rect.x, rect.y = x, y
- exception: TypeError if x, y are not of type Integer.

get_lives():

- output: out := self.lives
- exception: None

death_in_game():

- transition: if self.lives != 0 ⇒ self.restart, lives := True, lives - 1 *//If lives are not zero, then restart level.*
  else call game_over()
- exception: None

Local Types

None

Local Functions

None

Camera Module

Uses

None

Syntax

Exported Constants

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new Camera	$\mathbb{N}$, entity	Camera	
move			

Semantics

State Variables

pos: Object of type Vector2D *Contains the coordinates for camera position.*
entity: Object of type Entity
x: $\mathbb{N}$
y: $\mathbb{N}$

State Invariant

None

Assumptions & Design Decisions

- The Camera Constructor is called before any other access routines are called. Once called, the constructor will then not be called upon again.

Access Routine Semantics

new Camera(pos, entity):

- transition:
  self.pos := Vector2D(pos.x, pos.y)
  self.entity := entity
  self.x := pos.get_x()
  self.y := pos.get_y()

- exception: None

move():

- transition: x_pos_float := entity.get_pos_index_as_float().get_x(). if 10 < x_pos_float < 50 ⇒ pos := Vector2D(x_pos_float + 10, pos.get_y())
  x := pos.get_x() * 32
  y := pos.get_y() * 32

- exception None

Local Types

None

Local Functions

None

Level Module

Uses

None

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Level	Screen, Dashboard	Level	
load_level	String		FileNotFoundError
load_entities	JSON		
load_layers	JSON		
load_objects	JSON		
update_entities	Camera		
draw_level	Camera		IndexError
add_cloud_sprite	$\mathbb{N}, \mathbb{N}$		IndexError
add_pipe_sprite	$\mathbb{N}, \mathbb{N}, \mathbb{N}$		IndexError
add_bush_sprite	$\mathbb{N}, \mathbb{N}$		IndexError
add_random_box	$\mathbb{N}, \mathbb{N}$		
add_coin	$\mathbb{N}, \mathbb{N}$		
add_goomba	$\mathbb{N}, \mathbb{N}$		
add_koopa	$\mathbb{N}, \mathbb{N}$		

Semantics

State Variables

sprites: Object of type Sprite
dashboard: Object of type Dashboard
screen: Object of type Screen
level: Object of type Level
level_length: $\mathbb{N}$
entity_list: Seq of Entity

State Invariant

None

Assumptions & Design Decisions

- The Level constructor is called before any other access routines are called. Once called, the constructor will then not be called upon again.

Access Routine Semantics

new Level(screen, dashboard):

- transition:
  sprites := sprites()
  dashboard := dashboard
  screen := screen
  level := None
  level_length := 0
  entity_list := []

- exception: None

load_level(levelname):

- transition:
  data := open(levelname) as json_data ⇒ json.load(json_data)
  Call load_layers(data)
  Call load_objects(data)
  Call load_entities(data)
  level_length := data[”length”]

- exception: FileNotFoundError triggered if file is not found.

load_entities(data):

- transition:

$c =$ random_box	add_random_box(x, y) ⇒ $\forall x, y \in data[”level”][”entities”][c]$
$c =$ goomba	add_goomba(x, y) ⇒ $\forall x, y \in data[”level”][”entities”][c]$
$c =$ koopa	add_koopa(x, y) ⇒ $\forall x, y \in data[”level”][”entities”][c]$
$c =$ coin	add_coin _box(x, y) ⇒ $\forall x, y \in data[”level”][”entities”][c]$

- exception: None

load_layers(data):

- transition:
  layers := [ ] *//Initializes an empty sequence*
  $\forall x \in data[”level”][”layers”][”sky”][”x”] \mid (\forall y \in data[”level”][”layers”][”sky”][”y”]$: layers + Tile(sprites.sprite_collection.get(”sky”), None))
  $\forall x \in data[”level”][”layers”][”ground”][”x”] \mid (\forall y \in data[”level”][”layers”][”ground”][”y”]$: layers + Tile(sprites.sprite_collection.get(”ground”), None))
  //This is initializing the sky and ground blocks and appending them to a layer sequence.exception : *None*

load_objects(data):

- transition:

i = bush	add_bush_sprite(x, y) $\Rightarrow \forall x, y \in data[”level”][”objects”][c]$
i = cloud	add_cloud_sprite(x, y) $\Rightarrow \forall x, y \in data[”level”][”entities”][c]$
i = pipe	add_pipe_sprite(x, y) $\Rightarrow \forall x, y \in data[”level”][”entities”][c]$

- exception: None

update_entities(cam):

- transition: ∀ entity ∈ entity_list : entity.update(cam)∧

entity.alive	None
¬entity.alive	entity_list.remove(entity)

- exception: None

draw_level(camera):

- transition: ∀ y ∈ [0..15] : ∀ x ∈ [0−camera.pos.get_x()+1..20−camera.pos.get_x()−1]

level[y][x].sprite	level[y][x].sprite.redraw_background	screen.blit(sprite_collection.get(”sky”).image, (x + camera.pos.get_x()) * 32, y * 32) ∧ level[y][x].sprite.draw_sprite(x + camera.pos.get(x), y, screen)) ∧ update_entities(camera)
	¬level[y][x].sprite.redraw_background	level[y][x].sprite.draw_sprite(x + camera.pos.get(x), y, screen) ∧ update_entities(camera)
¬level[y][x].sprite		update_entities(camera)

- exception: IndexError if x, y are out of range.

add_cloud_sprite(x, y):

- transition: ∀ y_off ∈ [0..2] : (∀ x_off ∈ [0..3] : level[y + y_off][x + x_off] = Tile(sprites.sprite_collection.get(”cloud”, None))

- exception: IndexError if x, y are out of range.

add_pipe_sprite(x, y, length):

- transition:
  length := 2
  level[y][x] = Tile(sprites.sprite_collection.get(”pipeL”), pygame.Rect(x * 32, y * 32, 32, 32))
  level[y][x] = Tile(sprites.sprite_collection.get(”pipeR”), pygame.Rect(x * 32, y * 32, 32, 32))
  $\forall\ i \in (1, length + 20)$: level[y + i][x] = Tile(sprites.sprite_collection.get(”pipe2L”), pygame.Rect(x * 32, (y + i) * 32, 32, 32))
  $\forall\ i \in (1, length{+}20)$: level[y + i][x + 1] = Tile(sprites.sprite_collection.get(”pipe2R”), pygame.Rect((x + 1) * 32, (y + i) * 32, 32, 32))

- exception: IndexError if x, y are out of range.

add_bush_sprite(x, y):

- transition:
  level[y][x] = Tile(sprites.sprite_collection.get("bush_1"), None)
  level[y][x+1] = Tile(sprites.sprite_collection.get("bush_2"), None)
  level[y][x+2] = Tile(sprites.sprite_collection.get("bush_3"), None)

- exception: IndexError if x, y are out of range.

add_random_box(x, y):

- transition:
  level[y][x] = Tile(None, pygame.Rect(x * 32, y * 32 - 1, 32, 32))
  entity_list := entity_list + ⟨Random_Box(screen, sprites.sprite_collection, x, y, dashboard)⟩

- exception: None

add_coin(x, y):

- transition: entity_list := entity_list + ⟨Coin(screen, sprites.sprite_collection, x, y)⟩

- exception: None

add_goomba(x, y):

- transition: entity_list := entity_list +⟨Goomba(screen, sprites.sprite_collection, x, y, self)⟩

- exception: None

add_koopa(x, y):

- transition: entity_list := entity_list + ⟨Koopa(screen, sprites.sprite_collection, x, y, self)⟩

- exception: None

Local Types

None

Local Functions

None

Input Module

Uses

None

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Input	Entity_Base	Input	
check_for_input			
check_for_keyboard_input			
check_for_quit_and_restart_input_events			

Semantics

State Variables

mouse_X: $\mathbb{N}$
mouse_Y: $\mathbb{N}$
entity: Object of type Entity_Base

State Invariant

None

Assumptions & Design Decisions

- The Input constructor is called before any other access routines are called. Once called, the constructor will then not be called upon again.

Access Routine Semantics

new Input(entity):

- transition:
  mouse_X := 0
  mouse_Y := 0
  entity := entity
- exception: None

check_for_input():

- transition:
  Call check_for_keyboard_input()
  Call check_for_mouse_input()
  check_for_quit_and_restart_input_events()
- exception: None

check_for_keyboard_input():

- transition:
  pressed_keys := pygame.key.get_pressed()
  is_jumping := pressed_keys[K_SPACE] $\vee$ pressed_keys[K_UP]
  entity.traits["jumpTrait"].jump(is_Jumping) entity.traits["goTrait"].boost = pressed_keys[L_SHIFT]
  direction := entity.traits["goTrait"].direction

pressed_keys[K_LEFT] ∧ ¬ pressed_keys[K_RIGHT]	direction = -1
pressed_keys[K_RIGHT] ∧ ¬ pressed_keys[K_LEFT]	direction = 1
else	direction = 0

- exception: None

check_for_quit_and_restart_input_events():

- transition:
  events := pygame.event.get()
  ∀ event ∈ events — event.type == pygame.QUIT : pygame.quit() ∧ sys.exit()

  ∀ event ∈ events — event.type == pygame.KEYDOWN ∧ event.key == pygame.K_ESCAPE : entity.pause := True ∧ entity.pause_obj.create_background_blur()
- exception: None

Local Types

None

Local Functions

None

Vector2D Module

Uses

N/A

Syntax

Exported Types

Vector2D = tuple of (x: float, y: float)

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Vector2D	$\mathbb{R}, \mathbb{R}$	Vector2D	TypeError
get_x	—	$\mathbb{R}$	—
get_y	—	$\mathbb{R}$	—
add	Vector2D	—	TypeError
set_x	$\mathbb{R}$	—	TypeError
set_y	$\mathbb{R}$	—	TypeError
mag	—	$\mathbb{R}$	—

Semantics

State Variables

x: $\mathbb{R}$ // Represents the x component of the vector
y: $\mathbb{R}$ // Represents the y component of the vector

State Invariant

None

Assumptions & Design Decisions

None

Access Routine Semantics

new Vector2D(x, y):

- transition: $x, y := x, y$
- output: $out := self$
- exception: x, y not of type $\mathbb{R} \Rightarrow$ TypeError.

get_x():

- output: $out := x$

get_y():

- output: $out := y$

add(v):

- transition: $x, y := x + v.get\_x(), y := y + v.get\_y()$
- exception: v is not of type Vector2D $\Rightarrow$ TypeError

set_x(x):

- transition: $x := x$
- exception: x is not of type $\mathbb{R} \Rightarrow$ TypeError

set_y(y):

- transition: $y := y$
- exception: y is not of type $\mathbb{R} \Rightarrow$ TypeError

mag():

- output: $out := \sqrt{x^2 + y^2}$

Local Types

None

Local Functions

None

Sound_Controller Module

Uses

pygame.mixer.Channel	//	Contains methods for controlling a sound channel
pygame.mixer.Sound	//	Contains methods for loading sounds from a file

Syntax

Exported Types

N/A

Exported Constants

SOUNDTRACK	=	Main soundtrack
COIN_SOUND	=	Sound for collecting a coin
BUMP_SOUND	=	Sound when objects are bumped
STOMP_SOUND	=	Sound when Mario stomps an enemy
JUMP_SOUND	=	Sound when Mario jumps
DEATH_SOUND	=	Sound when Mario dies

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new sound_controller	—	Sound_Controller	—
play_sfx	Sound	—	TypeError
sfx_muted	—	$\mathbb{B}$	—
playing_sfx	—	$\mathbb{B}$	—
play_music	Sound	—	TypeError
music_muted	—	$\mathbb{B}$	—
playing_music	—	$\mathbb{B}$	—
stop_sfx	—	—	—
mute_sfx	—	—	—
unmute_sfx	—	—	—
stop_music	—	—	—
mute_music	—	—	—
unmute_music	—	—	—

Semantics

State Variables

music_ch: Channel // Channel over which music will be played
music_muted: $\mathbb{B}$ // Represents whether music can be played
sfx_ch: Channel // Channel over which sound effects will be played
sfx_muted: $\mathbb{B}$ // Represents whether sound effects can be played

State Invariant

None

Assumptions & Design Decisions

None

Access Routine Semantics

new Sound_Controller():

- transition:

  sfx_ch, music_ch := Channel(0), Channel(1)

  sfx_muted, music_muted := $False, False$

- output: *out* := *self*

play_sfx(s):

- transition: $\neg$ sfx_muted() $\Rightarrow$ play s over the sfx_ch channel
- exception: s not of type Sound $\Rightarrow$ TypeError

sfx_muted():

- output: *out* := sfx_muted

playing_sfx():

- output: *out* := sfx_ch.get_busy() // This method returns: $True$ if a sound is playing on the channel, $False$ otherwise.

play_music(s):

- transition: $\neg$ music_muted() $\Rightarrow$ play s over the music_ch channel
- exception: s not of type Sound $\Rightarrow$ TypeError

music_muted():

- output: out := music_muted

playing_music():

- output: out := music_ch.get_busy()

stop_sfx():

- transition: Call sfx_ch.stop(), which stops any sound playing on the sfx_ch channel

mute_sfx():

- transition: Call stop_sfx(), then set sfx_muted := $True$

unmute_sfx():

- transition: sfx_muted := $False$

stop_music():

- transition: Call music_ch.stop()

mute_music():

- transition: Call stop_music(), then set music_muted := $True$

unmute_music():

- transition: music_muted := $False$

Local Types

None

Local Functions

None

Spritesheet Module

Uses

pygame.Rect		
pygame.Surface	//	Class for representing images
pygame.image	//	Contains methods for loading images from files

Syntax

Exported Types

N/A

Exported Constants

N/A

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new Spritesheet	string	Spritesheet	—
image_at	$\mathbb{N}, \mathbb{N}, \mathbb{R}, (\mathbb{N}, \mathbb{N}, \mathbb{N}), \mathbb{B}, \mathbb{N}, \mathbb{N}$	Surface	TypeError

Semantics

State Variables

sheet: Surface // Represents an entire sheet of images in blocks

State Invariant

None

Assumptions & Design Decisions

None

Access Routine Semantics

new Spritesheet(filename):

- transition:

sheet := image.load(filename)

After assigning sheet, check if it has an alpha value in the pixels. If it does, then it is converted into a different pixel format while preserving the alpha, else it just converts the image.

- out: $out := self$

image_at$(x, y, scalingfactor, colorkey, ignoretilesize, xtilesize, ytilesize)$:

- out: This method creates a rectangle of the appropriate size (Rect$(x, y, xtilesize, ytilesize)$ or Rect$(x \cdot xtilesize, y \cdot ytilesize, xtilesize, ytilesize)$), then creates a surface from this rectangle. It then "cuts out" a portion of sheet of the rectangle size and copies it into the new surface. Lastly, the method returns an image that is scaled by the scalingfactor.

Local Types

None

Local Functions

None

Collider Module

Uses

Entity Level

Syntax

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new Collider	Entity, Level	Collider	—
check_x	—	—	
check_y	—	—	
right_level_border_reached	—	$\mathbb{B}$	—
left_level_border_reached	—	$\mathbb{B}$	—

Semantics

State Variables

entity: Entity	//	Entity to check collision for
level: list	//	list of objects to check for collidable objects
level_obj: Level	//	The level object itself

State Invariant

None

Assumptions & Design Decisions

None

Access Routine Semantics

new Collider(entity, level):

- transition: entity, level_obj, level := entity, level, level.level
- output: $out := self$

check_x():

- transition: Checks if entity is colliding with any level objects in the x direction. If so, it sets the entities horizontal velocity to 0, and updates the position of the entity so they are no longer colliding (if colliding on left, set x coordinate so that the objects are no longer intersecting).

check_y():

- transition: Checks if entity is colliding with any level objects in the y direction. If so, it sets the entities vertical velocity to 0, and updates the position of the entity so they are no longer colliding (if colliding on top, set y coordinate so that the objects are no longer intersecting).

right_level_border_reached():

- output: entity.x > level.level_length ⇒ True

left_level_border_reached():

- output: entity.x < 0 ⇒ True

Local Types

None

Local Functions

None

Animation Module

Uses

pygame.Surface

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
new Animation	List[Surface], Surface, Surface, $\mathbb{N}$	Animation	—
update	—	—	—
idle	—	—	—
in_air	—	—	—

Semantics

State Variables

images: List[Surface]	//	Contains the images to be part of the animation sequence
timer: $\mathbb{N}$	//	Keeps track of the time the animation has been going on
index: $\mathbb{N}$	//	Keeps track of the index of the current frame from images
image: Surface	//	The current image in the animation
idle_sprite: Surface	//	The default sprite when the animation is stopped
air_sprite: Surface	//	The default sprite when an entity is in the air
delta_time: $\mathbb{N}$	//	The time it takes for the animation to complete a cycle

State Invariant

None

Assumptions & Design Decisions

None

Access Routine Semantics

new Animation(images, idle_sprite, air_sprite, delta_time):

- transition:

    timer, index := 0, 0

images, image := images, images[index]

idle_sprite, air_sprite, delta_time := idle_sprite, air_sprite, delta_time

- output: $out := self$

update():

- transition:

timer := timer + 1

timer % delta_time = 0	$\text{index} < \lvert\text{images}\rvert - 1 \Rightarrow \text{index} := \text{index} + 1$
	$\neg\ \text{index} < \lvert\text{images}\rvert - 1 \Rightarrow \text{index} := 0$

image := images[index]

idle():

- transition: image := idle_sprite

in_air():

- transition: image := air_sprite

Local Types

None

Local Functions

None

Sprites Module

Uses

Spritesheet Animation pygame.Surface

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
new sprites	—	Sprites	—
load_sprites	Sequence[string]	Map[string:Surface — Animation]	—

Semantics

State Variables

sprite_collection: Map[string:Surface — Animation] // Contains the name of sprites mapped to their image

State Invariant

None

Assumptions & Design Decisions

None

Access Routine Semantics

new sprites():

- transition: Initialize sprite_collection by calling load_sprites with a list of file paths.
- output: $out := self$

load_sprites(file_paths):

- transition: Goes through each .json file (defined in file_paths) and parses them. Creates a Spritesheet object, and using information in the json file, it calls Spritesheet.image_at(...). It then updates res_dict, and maps the name from the .json file to the image it gets from Spritesheet.image_at(...). If the image is part of a sequence of images, then an Animation object is created with the sequence of images instead of a Surface.

Local Types

None

Local Functions

None

Sprite Module

Uses

Animation pygame.Surface

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
new sprite	Surface, $\mathbb{B}$, Animation, $\mathbb{B}$	Sprite	—
draw_sprite	$\mathbb{Z}, \mathbb{Z}$, Surface	—	—

Semantics

State Variables

image: Surface	//	Represents the sprite image
colliding: $\mathbb{B}$	//	Represents the collision state of the sprite
animation: Animation	//	Represents an animation object, if it is not None
redraw_background: $\mathbb{B}$	//	If true, redraw the background before drawing the sprite

State Invariant

None

Assumptions & Design Decisions

None

Access Routine Semantics

new sprite(image, colliding, animation, redraw_background):

- transition: image, colliding, animation, redraw_background := image, colliding, animation, redraw_background
- output: $out := self$

draw_sprite(x, y, screen):

- transition:

animation = None ⇒ screen.blit(image, 32 * x, 32 * y)

animation ≠ None ⇒ animation.update, screen.blit(animation.image, 32 * x, 32 * y)

Local Types

None

Local Functions

None

Menu Module

Template Module

Menu(screen, dashboard, level)

Uses

animation - spritesheet dashboard levels display - screen settings.json

Syntax

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new menu	screen, dashboard, level	menu	invalid_argument
update			
draw_dot			
load_settings	string		
save_settings	string		
draw_menu			
draw_menu_background			
draw_settings			
choose_level			
draw_border	$\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, set of $\mathbb{R}$, $\mathbb{N}$		
draw_level_chooser			
load_level_names		list of strings	
check_input			

Semantics

State Variables

screen : screen // from display module
start : $\mathbb{B}$
in_settings: $\mathbb{B}$
state: $\mathbb{N}$ // Represents where in menu user is
level: level // from level module
music : $\mathbb{B}$
sfx: $\mathbb{B}$

$current\_selected\_level$: $\mathbb{N}$ // defaults to first level
$level\_names$: []
$in\_choosing\_level$: $\mathbb{B}$
$dashboard$: dashboard // from dashboard module
$level\_count$: $\mathbb{N}$
$spritesheet$: $spritesheet$ from module Spritesheet
$menu\_banner$: obejct from $spritesheet$
$menu\_dot$: object from $spritesheet$
$menu\_dot2$: object from $spritesheet$

State Invariant

$spritesheet \neq$ None
$|level\_names| \geq current\_selected\_level$

Assumptions and Design Decisions

- None

Access Routine Semantics

menu($screen$, $dashboard$, $level$):

- transition:
  - $screen$:= screen
  - $start$: False
  - $in\_settings$: False
  - $state$: 0 // Represents where in menu user is
  - $level$: level // from level module
  - $music$: True
  - sfx: True
  - $current\_selected\_level$: 0
  - $level\_names$: []
  - $in\_choosing\_level$: False
  - $dashboard$: dashboard
  - $level\_count$: 0

  - *spritesheet* : Spritesheet("./resources/img/title_screen.png")
  - *menu_banner* :*spritesheet*.image_at(0, 60, 2, colorkey=[255, 0, 220], ignoreTileSize=True, xTileSize=180, yTileSize=88)
  - *menu_dot* : *spritesheet*.image_at(0, 150, 2, colorkey=[255, 0, 220], ignoreTileSize=True)
  - *menu_dot2* : *spritesheet*.image_at(20, 150, 2, colorkey=[255, 0, 220], ignoreTileSize=True)
  - load("./settings.json")

- exception: $exc := (screen \equiv None \vee dashboard \equiv None \vee level \equiv None) \Rightarrow$ invalid_argument)

update():

- transition: first check inputs using *check_input* before:

$in\_choosing\_level \equiv True$	exit
$in\_choosing\_level \equiv False$	*draw_menu_background*, update *dashboard*
$in\_choosing\_level \equiv False$ $\wedge$ in_settings $\equiv False$	*draw_menu*
$in\_choosing\_level \equiv False$ $\wedge$ in_settings $\equiv True$	*draw_settings*

- exception: None

draw_dot():

- transition:

$state \equiv 0$	*screen*.blit(*menu_dot*, (145, 273)) *screen*.blit(*menu_dot2*, (145, 313)) *screen*.blit(*menu_dot2*, (145, 353))
$state \equiv 1$	*screen*.blit(*menu_dot*, (145, 313)) *screen*.blit(*menu_dot2*, (145, 273)) *screen*.blit(*menu_dot2*, (145, 353))
$state \equiv 2$	*screen*.blit(*menu_dot*, (145, 353)) *screen*.blit(*menu_dot2*, (145, 273)) *screen*.blit(*menu_dot2*, (145, 313))

- exception: None

load_settings(string):

- transition: open *url* and use json.load to create required *data*

$data \equiv$ ”*sound*”	$music$ = True, $SOUND\_CONTROLLER$.unmute_music(), $SOUND\_CONTROLLER$.play_music($SOUNDTRACK$)
$data \neq$ ”*sound*”	$music$ = False, $SOUND\_CONTROLLER$.mute_music()
$data \equiv$ ”sfx”	sfx = True, $SOUND\_CONTROLLER$.unmute_sfx()
$data \neq$ ”sfx”	sfx = False, $SOUND\_CONTROLLER$.mute_sfx()

- exception: $IOError \lor OSError \Rightarrow music = False \land sfx = False \land$ $SOUND\_CONTROLLER.mute\_music() \land SOUND\_CONTROLLER.mute\_sfx() \land$ $save\_settings(\text{”}./settings.json\text{”})$

save_settings(string):

- transition: create a dictionary for *music* and sfx before using $json.dump$

- exception: None

draw_menu():

- transition:
  $draw\_dot()$
  The options ”CHOOSE LEVEL”, ”SETTINGS”, ”EXIT” are written on the dashboard.

- exception: None

draw_menu_background():

- transition:
  $(\forall y : \mathbb{N}|y \in [0..13] : \forall x : (\mathbb{N}|x \in [0..20] : screen.blit(self.level.sprites.spriteCollection.get(\text{”}sky\text{”}).image, (x * 32, y * 32)))$

  $(\forall y : \mathbb{N}|y \in [13..15] : \forall x : (\mathbb{N}|x \in [0..20] : screen.blit$ $(self.level.sprites.spriteCollection.get(\text{”}ground\text{”}).image, (x * 32, y * 32)))$

  Using the function *blit* from the module screen, the banner, mario and goomba icons and the bushes are placed on the menu background.

- exception: None

draw_settings():

- transition:
  $draw\_dot()$

  In the settings menu, writes using the dashboard method *draw_text* to write the words ”MUSIC”, ”SFX” and ”BACK” as well as:

$music \equiv True$	”ON”
$music \equiv False$	”OFF”
$sfx \equiv True$	”ON”
$sfx \equiv False$	”OFF”

- exception: None

choose_level():

- transition:
  $draw\_menu\_background(False)$ $\wedge$in$_chooosing_levels$

- exception: None

draw_level_chooser():

- transition: Using data from *load_level_names*, each level is titled and drawn as a button in the correct location in the menu.
- exception: None

load_level_names():

- output: Loads level names from the file in ”./resources/levels” and returns them into a list.
- transition: Updates *level_count* to equal the length of the created list.
- exception: None

check_input():

- transition: Uses *pygame.event* to collect all the user's inputs and place them into *events*, after which the type of event in sequence is funnelled into a state machine using a for statement composed of if statements:

event.type ≡pygame.QUIT	$pygame.quit(), sys.exit()$

$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_ESCAPE\ \wedge$ $(in\_choosing\_level \equiv True \vee in\_settings \equiv True$	$in\_choosing\_level = False, in\_settings = False$, re-initialize $screen, dashboard, level$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_ESCAPE\ \wedge$ $(in\_choosing\_level \equiv False \vee in\_settings \equiv False$	$pygame.quit(), sys.exit()$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_UP \wedge$ $(in\_choosing\_level \equiv True \wedge$ $current\_selected\_level > 3$	$current\_selected\_level- = 3, draw\_level\_chooser$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_UP \wedge$ $state > 0$	$state- = 1$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_DOWN \wedge$ $(in\_choosing\_level \equiv True \wedge$ $current\_selected\_level + 3 <= level\_count$	$current\_selected\_level+ = 3, draw\_level\_chooser$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_DOWN \wedge$ $state < 2$	$state+ = 1$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_LEFT \wedge$ $current\_selected\_level > 1$	$current\_selected\_level- = 1, draw\_level\_chooser$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_RIGHT \wedge$ $current\_selected\_level < level\_count$	$current\_selected\_level+ = 1, draw\_level\_chooser$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_RETURN \wedge$ $(in\_choosing\_level \equiv True$	$in\_choosing\_level = False, dashboard.state = "start"$, $dashboard.time = 420$, $level.load\_level(level\_names[$ $current\_selected\_level - 1])$, $dashboard.level\_name = level\_names[$ $current\_selected\_level - 1].split("Level")[1]$, $start = True$, EXIT
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_RETURN \wedge$ $(in\_settings \equiv False \wedge state \equiv 0$	$choose\_level()$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_RETURN \wedge$	$in\_settings = True, state = 0$

$(in\_settings \equiv False \wedge state \equiv 1$	
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_RETURN \wedge$ $(in\_settings \equiv False \wedge state \equiv 2$	$pygame.quit(), sys.exit()$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_RETURN \wedge$ $(in\_settings \equiv True \wedge state \equiv 0$ $\wedge music \equiv True$	$music = False,$ $SOUND\_CONTROLLER.stop\_music()$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_RETURN \wedge$ $(in\_settings \equiv True \wedge state \equiv 0$ $\wedge music \equiv False$	$music = TRUE,$ $SOUND\_CONTROLLER.play\_music($ $SOUNDTRACK)$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_RETURN \wedge$ $(in\_settings \equiv True \wedge state \equiv 0$	$save\_settings(\text{"}./settings.json\text{"})$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_RETURN \wedge$ $(in\_settings \equiv True \wedge state \equiv 1$ $\wedge sfx \equiv True$	$sfx = False, SOUND\_CONTROLLER.mute\_sfx()$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_RETURN \wedge$ $(in\_settings \equiv True \wedge state \equiv 1$ $\wedge sfx \equiv False$	$sfx = True, SOUND\_CONTROLLER.unmute\_sfx()$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_RETURN \wedge$ $(in\_settings \equiv True \wedge state \equiv 1$	$save\_settings(\text{"}./settings.json\text{"})$
$event.type \equiv$pygame.KEYDOWN$\wedge$ $event.key \equiv pygame.K\_RETURN \wedge$ $(in\_settings \equiv True \wedge state \equiv 2$	$in\_settings = False$

After the state machine runs through, and if it doesn't exit the method during execution, the display is updated using $pygame.display.update()$.

- exception: None

Local Types

None

Local Functions

None

Dashboard Module

Template Module

dashboard

0.1 Uses

display - screen Mario

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Dashboard	screen, $\mathbb{N}$	Dashboard	invalidArgument
update			
draw_text	string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$		
coin_string		string	
point_string		string	
time_string		string	

Semantics

State Variables

$state$: string
$screen$: instance of type screen
$level\_name$: string
$points$: $\mathbb{N}$
$coins$: $\mathbb{N}$
$ticks$: $\mathbb{N}$

time : $\mathbb{N}$
lives: $\mathbb{N}$

State Invariant

- $time \leq 420$
- $coins \geq 0$
- $points \geq 0$
- $1 \leq lives \leq 3$

Assumptions and Design Decisions

None

Access Routine Semantics

dashboard(screen, size):

- transition: $state$ = "menu"
  $screen$ = screen
  $level\_name$ = "" //empty string
  $points = 0$
  $coins = 0$
  $ticks = 0$
  $time = 420$
  $lives = Mario.get_lives()$

- exception: $exc := screen \equiv None \Rightarrow invalidArguemnt$

update():

- transition: Uses the methods *draw_text* to write the words "MARIO", "WORLD", "TIME" as well using $coin\_string, point\_string, time\_string$ to write the official values of coin, point and time. The official value of time is only written when $state \neq$ "menu". Upon the player losing a life, the *lives* state variable is updated by $lives = Mario.get_lives()$. Lastly, this method also updates the time value:

$True$	$ticks+ = 1$
$ticks \equiv 60$	$ticks = 0$, $time- = 1$

- exception: None

draw_text(text, x, y, size):

- transition: $(\forall char \in text : char\_sprite = pygame.transform.scale(FONT\_SPRITES[char], (size, size))$
  $screen.blit(char\_sprite, (x, y)) \wedge \begin{array}{|l|l|}\hline char \equiv ”\ ” & x+ = size//2 \\ \hline char \neq ”\ ” & x+ = size \\ \hline \end{array})$
- exception: None

coin_string():

- output: ”{:02d}”.format($coins$)
- exception: None

point_string():

- output: ”{:06d}”.format($points$)
- exception: None

time_string():

- output: ”{:03d}”.format($time$)
- exception: None

Local Types

None

Local Functions

None

Pause Module

Template Module

Pause

Uses

animation - spritesheet
dashboard
entity
display - screen
menu

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new Pause	$screen, entity, dashboard$	Pause	invalidArgument
update			
draw_dot			
check_input			
create_background_blur			

Semantics

State Variables

$screen$: instance of type screen
$entity$: instance of type entity
$dashboard$: instance of type dashboard

$state : \mathbb{N}$
$spritesheet$: value of $Spritesheet()$
$pause\_srfc$: value of $GaussianBlur()$
dot : instance of $spritesheet$
$gray\_dot$: instance of $spritesheet$

State Invariant

- $0 \leq state \leq 1$

Assumptions and Design Decisions

None

Access Routine Semantics

pause(screen, entity, dashboard):

- transition: $screen$: screen
  $entity$: entity
  $dashboard$: dashboard
  $state$: 0
  $spritesheet$: $Spritesheet(''./resources/img/title\_screen.png'')$
  $pause\_srfc$: $GaussianBlur().filter(screen, 0, 0, 640, 480)$
  dot: $spritesheet.image\_at(0, 150, 2, colorkey = [255, 0, 220], ignoreTileSize = True)$
  $gray\_dot$: $spritesheet.image\_at(20, 150, 2, colorkey = [255, 0, 220], ignoreTileSize = True)$

- exception: $exc := screen \equiv None \lor entity \equiv None \lor dashboard \equiv None \Rightarrow invalidArgument$

update():

- transition: Creates the pause menu over top of the game play screen using $pause\_srfc$ which blurs the background. The words "PAUSED", "CONTINUE" and "BACK TO MENU" are written on the screen, respectively top to bottom, and dots are placed to determine where the selector is.
- exception: None

draw_dot():

- transition:

$state \equiv 0$	*grey_dot* placed beside lower option, *dot* placed beside upper
$state \equiv 1 \equiv 60$	*grey_dot* placed beside upper option, *dot* placed beside upper

- exception: None

check_input():

- transition: Uses *pygame.event* to collect all the user's inputs and place them into *events*, after which the type of event in sequence is funnelled into a state machine using a for statement composed of if statements:

$event.type \equiv pygame.QUIT$	$pygame.quit(), sys.exit()$
$event.type \equiv pygame.KEYDOWN \wedge$ $event.key \equiv pygame.K\_RETURN \wedge$ $state \equiv 0$	$entity.pause = False$ $*$
$event.type \equiv pygame.KEYDOWN \wedge$ $event.key \equiv pygame.K\_RETURN \wedge$ $state \equiv 1$	$entity.restart = True$
$event.type \equiv pygame.KEYDOWN \wedge$ $event.key \equiv pygame.K\_UP \wedge$ $state > 0$	$state- = 1$
$event.type \equiv pygame.KEYDOWN \wedge$ $event.key \equiv pygame.K\_DOWN \wedge$ $state < 1$	$state+ = 1$

- exception: None

create_background_blur():

- transition: $pause\_srfc = GaussianBlur().filter(self.screen, 0, 0, 640, 480)$

- exception: None

Local Types

None

Local Functions

None

levels.json Module

Template Module

levels.json

Description

This is a document that contains the outlines of where different entities such as ground blocks or item boxes or sky etc. will be placed for a given level. This document is used to create the levels upon level initialization and menu initialization.