Staroids, Module Interface Specification

Team 20, Staroids
Moziah San Vicente, 400091284, sanvicem
Eoin Lynagh, 400067675, lynaghe
Jason Nagy, 400055130, nagyj2

November 9, 2018

The following is a series of MISes for the modules that comprise the Staroids game

Table 1: **Revision History**

Date	**Version**	**Notes**
Nov 06/18	0.1	Added basic information to template
Nov 07/18	0.2	Added Head module specification
Nov 08/18	0.3	Added all module specifications
Nov 09/18	0.35	Tidied up
Nov 09/18	0.5	Finished Sound, Utilities, Head and Game State MIS

Utilities Module

Template Module

Utilities

Uses

CVS from Browser (Playing screen)
CTX from CVS (Screen coordinate system)
FONTSTYLE from Browser (Available fonts for printing)

Syntax

Exported Types

FPS=30
SHIP_SIZE=30
TURN_SPEED=180
SHIP_THRUST=0.2
SHIP_BREAK=0.98
MIN_SPEED=0.1
MAX_SPEED=20
MAX_ACC=2
CVS_WIDTH=780
CVS_HEIGHT=620
BULLET_EXTRA=5
KILLABLE={True,False}
MAX_ASTEROIDS=2
TEST={True,False}
ALIEN_SPAWN=700
KeyCode={UP,DOWN,RIGHT,LEFT,SPACE,M,P,R}
EPOCH=1
Key=?
Text=?
Game=?

Exported Access Programs

Routine name	In	Out	Exceptions
Key		Key	
isDown	KeyCode	$\mathbb{N}$	
onKeydown	KeyCode	$\mathbb{N}$	
onKeyup	KeyCode		

Semantics

State Variables

d: sequence of $\mathbb{N}$

State Invariant

$\forall(c : \mathbb{N} | c \in d : c > 0)$

Assumptions

- Only known keys (as defined by KeyCode) will be put into the Key object as events to be processed.

Access Routine Semantics

Key():

- transition: $d :=$ seq of KeyCode
- output: $out := Key$
- exception: None

isDown(e):

- output: $e \in d \Rightarrow true \wedge e \notin d \Rightarrow false$
- exception: None

onKeydown(e):

- transition: $d[e] =$ EPOCH

- exception: None

onKeyup(e):

- output: $out := d[e]$
- exception: None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
TEXT	CTX, FONTSTYLE	TEXT	
norm	$String, \mathbb{Z}, \mathbb{Z}$		
emph	$String, \mathbb{Z}, \mathbb{Z}$		

Semantics

State Variables

cvs: CTX fnt: FONTSTYLE

State Invariant

None

Assumptions

- Before the Text object is used, the initialization function must be run first.

Access Routine Semantics

norm(Str, x, y):

- transition: Displays Str to cvs at location (x, y) in standard font.
- exception: None

emph(Str, x, y):

- transition: Displays Str to cvs at location (x, y) in emphasized font.
- exception: None

Routine name	**In**	**Out**	**Exceptions**
Game			
addScore	$\mathbb{Z}$		
addSprites	OBJECT		
subLives	$\mathbb{Z}$		
subSprites	OBJECT		
getScore		$\mathbb{N}$	
getLives		$\mathbb{N}$	
getLevel		$\mathbb{N}$	
getAsteroids		$\mathbb{N}$	
getWidth		$\mathbb{N}$	
getHeight		$\mathbb{N}$	
getCvs		CVS	
getCtx		CTX	
getSprites		sequence of OBJECT	
getPlayer		PLAYER	
getAlien		ALIEN	
getText		TEXT	
getSound		SOUND	
getPaused		$\mathbb{B}$	
setScore	$\mathbb{N}$		
setLives	$\mathbb{N}$		
setLevel	$\mathbb{N}$		
setAsteroids	$\mathbb{N}$		
setWidth	$\mathbb{N}$		
setHeight	$\mathbb{N}$		
setCvs	CVS		
setCtx	CTX		
setSprites	sequence of OBJECT		
setPlayer	PLAYER		
setAlien	ALIEN		
setText	TEXT		
setSound	SOUND		
setPaused	$\mathbb{B}$		
reduceCounter	$String, \mathbb{Z}, \mathbb{Z}$		
resetMute			
resetPause			
drawLives			

Semantics

State Variables

score: $\mathbb{N}$
lives: $\mathbb{N}$
level: $\mathbb{N}$
asteroids: $\mathbb{N}$
width: $\mathbb{N}$
height: $\mathbb{N}$
cvs: CVS
ctx: CTX
sprites: sequence of OBJECT
player: PLAYER
alien: ALIEN
text: TEXT
sound: SOUND
paused: $\mathbb{B}$
muteSound: $\mathbb{N}$
pauseGame: $\mathbb{N}$

State Invariant

None

Assumptions

None

Access Routine Semantics

Game():

- transition: $score = 0 \land lives = 3 \land sprites = seq.of\text{OBJECT} \land muteSound = FPS \land pauseGame = FPS$
- exception: None

getScore():

- output: $out := score$

- exception: None

getLives():

- output: $out := lives$
- exception: None

getLevel():

- output: $out := level$
- exception: None

getAsteroids():

- output: $out := asteroids$
- exception: None

getWidth():

- output: $out := width$
- exception: None

getHeight():

- output: $out := height$
- exception: None

getCvs():

- output: $out := cvs$
- exception: None

getCtx():

- output: $out := ctx$
- exception: None

getSprites():

- output: $out := sprites$

- exception: None

getPlayer():

- output: $out := player$
- exception: None

getAlien():

- output: $out := alien$
- exception: None

getText():

- output: $out := text$
- exception: None

getSound():

- output: $out := sound$
- exception: None

getPaused():

- output: $out := paused$
- exception: None

setScore(s):

- transition: $score = s$
- exception: None

setLives(l):

- transition: $lives = l$
- exception: None

setLevel(l):

- transition: $level = l$

- exception: None

setAsteroids(a):

- transition: $asteroids = a$
- exception: None

setWidth(w):

- transition: $width = w$
- exception: None

getHeight(h):

- transition: $height = h$
- exception: None

setCvs(c):

- transition: $cvs = c$
- exception: None

setCtx(c):

- transition: $cyx = c$
- exception: None

setSprites(s):

- transition: $sprites = s$
- exception: None

setPlayer(p):

- transition: $player = p$
- exception: None

setAlien():

- transition: $alien = a$

- exception: None

setText(t):

- transition: $text = t$
- exception: None

setSound(s):

- transition: $sound = s$
- exception: None

setPaused(b):

- transition: $paused = b$
- exception: None

reduceCounter():

- transition: $muteSound := muteSound - 1 \wedge pauseGame := pauseGame - 1$
- exception: None

resetMute():

- transition: $muteSound = FPS$
- exception: None

resetPause():

- transition: $pauseGame = FPS$
- exception: None

drawLives():

- transition: Draws *lives* amount of triangular ships to the top left corner of screen to represent player amount of lives left.
- exception: None

addScore(amount):

- transition: $score = score + amount$
- exception: None

addSprite(obj):

- transition: $sprites = sprites||obj$
- exception: None

subLives(obj):

- transition: $lives = lives - 1$
- exception: None

subSprite(obj):

- transition: $sprites = sprites \setminus obj$
- exception: None

Sound Module

Uses

AUDIO from .wav sound files
Within the AUDIO class, there are specific sounds:

- LASER for shooting projectiles
- BRAKE for the player ship braking
- EXPLOSION for the destruction

Syntax

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
Sound		AUDIO	
play	AUDIO		
isPlay	AUDIO	$\mathbb{B}$	
pause	AUDIO		
unpause	AUDIO		
stop	AUDIO		
mute			
unmute			
toggle			

Semantics

State Variables

$sounds = \{LASER, BRAKE, EXPLOSION\}$ $muted = \mathbb{B}$

State Invariant

None

Assumptions

- The constructor is called before other accesses
- The sound files are in the correct directory for the projectiles
- The sound files have the same name as expected.

Access Routine Semantics

Sound():

- transition: $muted := false$
- exception: None

play(x):

- input: $x \in$ sound
- transition: $\neg muted \Rightarrow Play(x)$
- exception: None

isPlay(x):

- input: $x \in$ sound
- output: Boolean to whether sound x is playing or not
- exception: None

pause(x):

- input: $x \in$ sound
- transition: $pauseSound(x)$
- exception: None

unpause(x):

- input: $x \in$ sound
- transition: $unpauseSound(x)$
- exception: None

stop(x):

- input: $x \in$ sound
- transition: $stopSound(x)$
- exception: None

mute(x):

- input: $x \in$ sound
- transition: $muted := true$
- exception: None

unmute(x):

- input: $x \in$ Sound
- transition: $muted := false$
- exception: None

toggle():

- transition: $muted = true \Rightarrow muted := false \vee muted = false \Rightarrow muted := true$
- exception: None

Head Module

Uses

FILE from modules (Takes the source file):

- Utilities
- Sound
- GameObject
- GameState

Exported Constants

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
dynamicallyLoadScript	FILE		

Semantics

State Variables

None

State Invariant

None

Assumptions

- The files are named the same way that the module expects

Access Routine Semantics

dynamicallyLoadScript(x):

- input: $x \in$ FILE
- transition: Appends x to the current file
- output: $out := Head$
- exception: None

GameObject Module

Template Module

GameObject

Uses

CVS from Browser (Playing screen)
CTX from CVS (Screen coordinate system)
Utilities
Sound

Syntax

Exported Types

GameObject=? Player=? Bullet=? Alien=? AlienBullet=? Asteroid=?

Exported Constants

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
GameObject		GameObject	
getX		$\mathbb{Z}$	
getY		$\mathbb{Z}$	
getHeading		$\mathbb{R}$	
getActivity		$\mathbb{B}$	
getRadius		$\mathbb{Z}$	
getVel		$\mathbb{R}$	
getCtx		CTX	
getName		String	
setX	$\mathbb{Z}$		
setY	$\mathbb{Z}$		
setActivity	$\mathbb{B}$		

Semantics

State Variables

name: String
x: $\mathbb{R}$
y: $\mathbb{R}$
rot: $\mathbb{R}$
a: $\mathbb{R}$
r: $\mathbb{N}$
visible: $\mathbb{B}$
vel: sequence of $\mathbb{R}$
acc: sequence of $\mathbb{R}$
ctx: CTX

State Invariant

None

Assumptions

GameObject(name):

- transition: $name, x, y, rot, a, visible, vel, acc, r, ctx = \text{name}, 0, 0, 0, 0, false, (0, 0), (0, 0), 0, \text{CTX}$
- output: $out := GameObject$
- exception: None

getX():

- output: $out := x$
- exception: None

getY():

- output: $out := y$
- exception: None

getHeading():

- output: $out := a$

- exception: None

getActivity():

- output: $out := visible$
- exception: None

getRadius():

- output: $out := r$
- exception: None

getVel():

- output: $out := vel$
- exception: None

getAcc():

- output: $out := acc$
- exception: None

getCtx():

- output: $out := ctx$
- exception: None

getName():

- output: $out := name$
- exception: None

setX(x):

- input: $in := x \in \mathbb{Z}$
- exception: None

setY(y):

- input: $in := x \in \mathbb{Z}$
- exception: None

setActivity(activity):

- input: $in := x \in \mathbb{B}$
- exception: None

Routine name	**In**	**Out**	**Exceptions**
Player		Player	
interact	KeyCode		
move			
draw		CVS	
action			
collide			
collideOffshoot			
die			
reset			

Semantics

State Variables

None

State Invariant

None

Assumptions

Player():

- inheret: GameObject $\Rightarrow$ All state variables and methods
- transition: $fire, thrust, turn, airbrake, bulletCountDownvel(x, y), acc(x, y), r = false, false, false,$
- output: $out := Player$
- exception: None

interact():

- input: $in := up \vee space \vee left \vee right \vee down \in$ KeyCode
- transition: $up, space, left, right, down := thrust = true, fire = true, turn = left, turn = right, airbrake = true$
- exception: None

move():

- input: $thrust, turn := true \vee false, left \vee right$
- transition: $thrust = true \Rightarrow acc.x+ =$ SHIP_THRUST $* cos(a)/$FPS $\wedge acc.y+ =$ SHIP_THRUST $* sin(a)/$FPS $\wedge vel.x+ = acc.x \wedge vel.y+ = acc.y, thrust = false \Rightarrow$ DO NOTHING$, turn = right \Rightarrow rot = -$TURN_SPEED$/180 *$ PI/FPS$, turn = left \Rightarrow rot =$ TURN_SPEED$/180*$PI/FPS$, turn \neg(right \vee left) \Rightarrow rot = 0, space, left, right, down :=$ $thrust = true, fire = true, turn = left, turn = right, airbrake = true$
- exception: $vel.x >=$ MAX_SPEED $\Rightarrow$ DO NOT TRANSITION THRUST AND DECREMENT VE $max \Rightarrow$ DO NOT TRANSITION THRUST AND DECREMENT VELOCITY UNTIL IT IS BELO

draw():

- input: Player
- transition: draws shape of player ship onto canvas including a thruster image if the ship is being thrusted.
- exception: None

action():

- input: fire and bullet countdown
- transition: if fire is set to true then a new bullet object is created the bullet sound is played and the and the bullet countdown is set to FPS/1.25. The bullet is also added to the sprite array, and the player object is passed through to the bullet in order for it to get its releative velocity and location from.
- exception: None

collide():

- input: none
- transition: Checks the spritearray from Game in utilities module to see if any asteroid, alien, or alienBullet objects are overlapping areas with the player and if so will kill the player.
- exception: None

collideOffshoot():

- input: none

- transition: Same as collide but recursively goes through the asteroids children to check them as well.
- exception: None

die():

- input: none
- transition: when player dies due to collision the game lives are decremented by one, the player is deactivated and the vel and acc in both the x and y directions are set back to zero.
- exception: None

reset():

- input: none
- transition: resets player flags back to original values: fire = false, thrust = false, turn = false, bulletCountdown -= 1, airbrake = false.
- exception: None

Routine name	**In**	**Out**	**Exceptions**
Bullet	Player	Bullet	

Semantics

State Variables

None

State Invariant

None

Assumptions

Bullet(p):

- inheret: GameObject $\Rightarrow$ All state variables and methods
- transition: $timeOut, vel, x, y, r, velx, vely = 200, \{\}, getX(p)+4/3*getR(p)*cos(getHeading(p)), get$ $4/3*getR(p)*sin(getHeading(p)), 1, getVelX(p)+\text{BULLET\_EXTRA}*cos(getHeading(p)), getVelY($ $\text{BULLET\_EXTRA} * -sin(getHeading(p))$
- output: $out := Bullet$
- exception: None

action():

- transition: $if timeOut <= 0 \Rightarrow this.deactivate() \wedge Game.subSprites(this) else timeOut := timeOut - 1$

move()

- transition: $this.x := this.x + this.vel.x \wedge this.y := this.y + this.vel.y$
- transition2: $(this.x < 0 \Rightarrow this.x = CVS\_WIDTH) \vee (this.y < 0 \Rightarrow this.y = CVS\_HEIGHT) \vee (this.x > CVS\_WIDTH \Rightarrow this.x = 0) \vee (this.y > CVS\_HEIGHT \Rightarrow this.y = 0)$

draw():

- transition: if the sprite is active, it draws a circle with radius 1 at the x and y location of the bullet.

collide():

- transition: if both the sprite its colliding with and itself are active, then if they collide, and the other object is an alien or asteroid, it destroys the asteroid/alien and the bullet then increases score.

collideOffshoot():

- transition: recursive version of collide for checking that its not colliding with asteroid children.

die():

- transition: $this.deactivate()$

getTimeout():

- output $= out := this.timeOut$

setTimeout(life):

- input $= in := life \in \mathbb{Z}$
- transition: $this.timeOut := life$

Routine name	**In**	**Out**	**Exceptions**
Alien	CTX	Alien	
draw			
move			
action			
collide			
collideOffshoot			
die			

Semantics

State Variables

All GameObject state variables $timeSpawn$: $\mathbb{N}$ $timeOut$: $\mathbb{N}$ $xOrY$: $\mathbb{B}$ $lOrR$: $\mathbb{B}$ acc: sequence of $\mathbb{N}$

State Invariant

None

Assumptions

- The constructor is called before any other Alien method is called.

Alien():

- inheret: GameObject $\Rightarrow$ All state variables and methods
- transition: $timeSpawn, timeOut, xOrY, lOrR, acc, r = \text{ALIEN\_SPAWN}, 50, true, true, (0,0), 12.5$
- output: $out := Alien$
- exception: None

draw():

- transition: If $visible = true$, it draws a square at (x, y) every frame to ctx.
- exception: None

move():

- transition: vel is added to x and y. This moves the Alien on the screen.

- exception: None

action():

- transition: The alien counts down until another AlienBullet is fired. *vel* is also adjusted to create a sinodial path for the Alien to move through.
- exception: None

collide():

- transition: Detects if a PLAYER, BULLET, or ASTEROID is within r pixels of the alien. If so, the alien executes die()
- exception: None

collideOffshoot(x):

- input: x = sequence of OBJECT
- transition: Detects if a PLAYER, BULLET, or ASTEROID is within r pixels of the alien. If so, the alien executes die(). If any object has children, collideOffshoot() is called again with those children as x
- exception: None

die():

- transition: Makes the Alien invisible and randomizes its location on *cvs*. *timeSpawn* is reset to ALIEN_SPAWN, *timeOut* to 50, *xOrY* and *lOrR* to 1 or 0 and true or false randomly, respectfully.
- exception: None

Routine name	**In**	**Out**	**Exceptions**
AlienBullet		AlienBullet	

Semantics

State Variables

None

State Invariant

None

Assumptions

AlienBullet(a):

- inheret: GameObject $\Rightarrow$ All state variables and methods
- transition: $timeOut, vel, x, y, r, velx, vely = 200, , getX(a), getY(a), 2, random(-3..3), numbersothat$ $|velY|^2 = 3$
- output: $out := AlienBullet$
- exception: None

action():

- transition: $iftimeOut <= 0 \Rightarrow this.deactivate() \wedge Game.subSprites(this) elsetimeOut := timeOut - 1$

move()

- transition: $this.x := this.x + this.vel.x \wedge this.y := this.y + this.vel.y$
- transition2: $(this.x < 0 \Rightarrow this.x = CVS\_WIDTH) \vee (this.y < 0 \Rightarrow this.y = CVS\_HEIGHT) \vee (this.x > CVS\_WIDTH \Rightarrow this.x = 0) \vee (this.y > CVS_HEIGHT \Rightarrow this.y = 0)$

draw():

- transition: if the sprite is active, it draws a red circle with radius 2 at the x and y location of the bullet.

collide():

- transition: if both the sprite its colliding with and itself are active, then if they collide, and the other object is a player or asteroid, it destroys the player/asteroid and the bullet.

collideOffshoot():

- transition: recursive version of collide for checking that its not colliding with asteroid children.

die():

- transition: $this.deactivate()$

getTimeout():

- output = $out := this.timeOut$

setTimeout(life):

- input = $in := life \in \mathbb{Z}$
- transition: $this.timeOut := life$

Routine name	**In**	**Out**	**Exceptions**
Asteroid		Asteroid	

Semantics

Uses

The JavaScript Math library for random, round and other functions.

State Variables

x: y: $scale$: vel: acc: $children$:

State Invariant

None

Assumptions

Asteroid():

- inheret: GameObject $\Rightarrow$ All state variables and methods
- input: $ctx = CTX, scale \in \mathbb{Z}$
- transition: $x, y, scale, r, children, vel, velx, vely = random(0...CVS\_WIDTH), random(0...CVS\_H$ $scale, [], random(-1...1) * 3, random(-1...1) * 3$
- output: $out := Asteroid$
- exception: None

draw():

- transition: If $visible = true$, it draws a circle at (x, y) every frame to ctx. If $visible = false$, it draws all asteroids in $children$

move():

- transition: $this.x := this.x + this.vel.x \wedge this.y := this.y + this.vel.y$
- transition2: $(this.x < 0 \Rightarrow this.x = CVS\_WIDTH) \vee (this.y < 0 \Rightarrow this.y = CVS\_HEIGHT) \vee (this.x > CVS\_WIDTH \Rightarrow this.x = 0) \vee (this.y > CVS\_HEIGHT \Rightarrow this.y = 0)$

action():

- transition: for testing, it checks if keys are being pressed each frame, and if they are, asteroids are destroyed, corresponding to the test key.

die():

- transition: calls the deactivate function, then if the asteroid is not small it creates 3 new smaller asteroids, and places them at its center.

pass():

- transition: updates all of the asteroids children, if they have all been destroyed then it removes the children from the game.

isDead():

- output: false if the asteroid is visible or has children left, or true if the asteroid has no children left or are all dead.

getChildren():

- output = $out := this.children$

getScale():

- output = $out := this.scale$

setChildren(children):

- input = $in = children \in GameObject[]$
- transition = $this.children := children$

setScale(scale):

- input = $in = scale \in \mathbb{Z}$
- transition = $this.scale := scale$

add(children):

- input = $in = children \in GameObject$
- transition = $this.children || children$

GameState Module

Uses

utilities.js, gameobject.js, head.js

Exported Constants

STATE={START,PREGAME,LOAD,PLAYING,POSTGAME,PAUSE,RELOAD}
StateMachine=?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
StateMachine			
isSafe	OBJECT, seq. of OBJECT		
generateAsteroids	$x \in \mathbb{Z}$		
checkCollision	GameObject, GameObject, GameObject		
togglePause			

Semantics

State Variables

state: String *stateSave*: String *paused*: $\mathbb{B}$

State Invariant

$state \neq stateSave$

Assumptions

None

Access Routine Semantics

StateMachine():

- transition: $state =$ start

- output: $out := StateMachine$
- exception: None

isSafe(obj,sprites)

- input: $in := object, in := sprites$
- return: $d := \forall s \in sprites : (getName(s) = "asteroid" \land getActivity(s) = false \land \exists c \in getChildren(s) : \neg isSafe(c) : false) \lor (getName(s) = "asteroid" \land getActivity(s) = true : checkCollision(obj, s, 50) : false) \lor (getName(s) \in \{"alien," alienBullet"\} \land getActivity(s) = true \land checkCollision(obj, s, 50) : false)$

checkCollision(a,b,c)

- input: $a \in \text{GameObject}, b \in \text{GameObject}, c \in \mathbb{Z}$
- output: $out := (pyth(|a.getX() - b.getX()|, |a.getX() - b.getX()|) < c)$

togglePause():

- transition: $pause \Rightarrow (stateSave = state \land state = \text{PAUSE}) \lor \neg pause \Rightarrow (state = stateSave)$
- exception: None

Local Functions

screenShow: $String \times \{\text{NORMAL}, \text{EMPHASIS} \Rightarrow ?\}$ output: $out :=$
drawShape: $String \times \mathbb{R} \times \mathbb{R} \Rightarrow$ transition: displays specifed shape at specified position
Play: $AUDIO$ transition: Plays specified AUDIO audio file
pauseSound: $AUDIO$ transition: Pauses specified AUDIO audio file
unpauseSound: $AUDIO$ transition: Unpauses specified AUDIO audio file
stopSound: $AUDIO$ transition: Stops specified AUDIO audio file is it was playing
drawTriangle: $\mathbb{N} \Rightarrow$ output: $out :=$
getName: OBJECT $\Rightarrow$ output: $out :=$
getActivity: OBJECT $\Rightarrow$ output: $out :=$
getChildren: OBJECT $\Rightarrow$ output: $out :=$