Staroids, Module Interface Specification

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

November 8, 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

Utilities Module

Template Module

Utilities

Uses

N/A

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

$\forall(c : \mathbb{N} | c \in d : c0)$

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	Screen, Font	Text	
norm	$String, \mathbb{Z}, \mathbb{Z}$		
emph	$String, \mathbb{Z}, \mathbb{Z}$		

Semantics

State Variables

cvs: Screen fnt: Font

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: $cvs[x][y] = screenShow(Str, \text{NORMAL})$
- exception: None

emph(Str, x, y):

- transition: $cvs[x][y] = screenShow(Str, \text{EMPHASIS})$
- exception: None

Routine name	**In**	**Out**	**Exceptions**
Game			
reduceCounter	$String, \mathbb{Z}, \mathbb{Z}$		
resetMute			
resetPause			
drawLives			
addScore	$\mathbb{Z}$		
addSprites	OBJECT		
subLives	$\mathbb{Z}$		
subSprites	OBJECT		

Semantics

State Variables

$score$: $\mathbb{N}$ $lives$: $\mathbb{N}$ $sprites$: sequence of OBJECT $muteSound$: $\mathbb{N}$ $pauseGame$: $\mathbb{N}$

State Invariant

None

Assumptions

None

Access Routine Semantics

Game():

- transition: $score = 0 \wedge lives = 3 \wedge sprites = seq.of\text{OBJECT} \wedge muteSound = FPS \wedge pauseGame = FPS$
- 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: $\forall(i : \mathbb{N} | i < lives : drawTriangle(i * 15))$
- exception: None

addScore(amount):

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

addSprite(obj):

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

subLives(obj):

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

subSprite(obj):

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

Sound Module

Uses

AUDIO for Sound

Syntax

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
Sound		Sound	
play	Sound		
isPlay	Sound	Boolean	
pause	Sound		
unpause	Sound		
stop	Sound		
mute			
unmute			
toggle			

Semantics

State Variables

Sound: Audio object from file

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: $muted := true$
- exception: None

play():

- input: $in := x \in$ Sound
- transition: $!in.muted : in.play()$
- exception: None

isPlay():

- input: $in := x \in$ Sound
- output: $out :=!in.paused()$
- exception: None

pause():

- input: $in := x \in$ Sound
- transition: $in.paused := true$
- exception: None

unpause():

- input: $in := x \in$ Sound
- transition: $in.paused :=!true$
- exception: None

stop():

- input: $in := x \in$ Sound
- transition: $in.paused := truethis.currentTime := 0$
- exception: None

mute():

- input: $in := x \in \text{Sound}$
- transition: $in.muted := true$
- exception: None

unmute():

- input: $in := x \in \text{Sound}$
- transition: $in.muted :=!true$
- exception: None

toggle():

- input: $in := x \in \text{Sound}$
- transition: $in.muted :=!in.muted$
- exception: None

Head Module

Uses

utilities.js, sound.js, gameobject.js, gamestate.js

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
dynamicallyLoadScript	any		

Semantics

State Variables

None

State Invariant

None

Assumptions

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

Access Routine Semantics

dynamicallyLoadScript():

- input: $in := x \in \{"utilities.js", "sound.js", "gameobject.js", "gamestate.js"\}$
- transition: $c := \{\}$
- output: $out := Head$
- exception: None

GameObject Module

Template Module

GameObject

Uses

N/A

Syntax

Exported Types

GameObject, Player, Bullet, Alien, AlienBullet, Asteroid

Exported Constants

None

Exported Access Programs

Routine name	**In** GameObject	**Out**	heightGameObject
update			
activate			
deactivate			
die			
interact			
move			
action			
draw			
reset			
pass			
collide			
update			
collide			
collide			

GameObject():

- transition: *state* = start

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

Routine name	**In**	**Out**	heightPlayer
	Player		
fire		Player	
thrust turn height			

Player():

- transition: $fire, thrust, turn, airbrake, bulletCountDownvel, acc, r =$ false, false, false, FPS/2, (0,0), (0,0

Routine name	**In**	**Out**	heightBullet
	Bullet		

Bullet():

- transition: $vel, x, y, r, velx, vely, , thrust, turn, airbrake, bulletCountDownvel, acc, r =$ false, false, false, F

Routine name	**In**	**Out**	heightAlien
	Alien		

Alien():

- transition: $fire, thrust, turn, airbrake, bulletCountDownvel, acc, r =$ false, false, false, FPS/2, (0,0), (0,0

Routine name	**In**	**Out**	heightAlienBullet
	AlienBullet		

AlienBullet():

- transition: $fire, thrust, turn, airbrake, bulletCountDownvel, acc, r =$ false, false, false, FPS/2, (0,0), (0,0

Routine name	**In**	**Out**	heightAsteroid
	Asteroid		

Asteroid():

- transition: $fire, thrust, turn, airbrake, bulletCountDownvel, acc, r =$ false, false, false, FPS/2, (0,0), (0,0

Semantics

State Variables

col: sequence of StackT
fre: sequence of CardT
fou: sequence of CardT
dek: DeckT

State Invariant

- All StackTs within *col* must have a CardT with getSuit()=NAS and getRank()=NAR at the bottom (first added on).

Assumptions

- The constructor BoardT is called for each object instance before any other access routine is called for that object. The constuctor cannot be called on an existing object.
- Unallocated fre locations are to be filled with a CardT with getSuit()=NAS and getRank()=NAR.

Access Routine Semantics

BoardT():

- transition: $col := \forall(c : \text{CardT}|c \in dek : col||c)$

  $fre :=$ seq of CardT

  $fou :=$ seq of CardT

  $dek :=$ DeckT()
- output: $out := self$
- exception: None

hasWon():

- output: $out := \text{BoardEmpty}(col) \wedge \forall(c : \text{CardT}|c \in fre : \text{FreeCellEmpty}(c)) \wedge forall(C : \text{CardT}|C \in fou : \text{FoundationComplete}(C))$
- exception: None

getStack(i):

- output: $out := col[i]$
- exception: $(\neg(0 <= i < 8) \Rightarrow \text{invalid\_index})$

getFree(i):

- output: $out := fre[i]$

- exception: $(\neg(0 <= i < 4) \Rightarrow \text{invalid\_index})$

getWin(i):

- output: $out := fou[i]$
- exception: $(\neg(0 <= i < 4) \Rightarrow \text{invalid\_index})$

setStack(i,S):

- transition: $col[i] = S$
- exception: $(\neg(0 <= i < 8) \Rightarrow \text{invalid\_index})$

getFree(i,C):

- transition: $fre[i] = C$
- exception: $(\neg(0 <= i < 4) \Rightarrow \text{invalid\_index})$

getWin(i,C):

- transition: $fou[i] = C$
- exception: $(\neg(0 <= i < 4) \Rightarrow \text{invalid\_index})$

moveColToCol(a,b):

- transition: $col[a], col[b] := col[a].\text{remCard}(), col[b].\text{addCard}(col[a].\text{peek}())$
- exception: $((\neg\text{ValidIndex}(8, 8, a, b) \Rightarrow \text{invalid\_index}) \vee (\text{StackEmpty}(col[a]) \Rightarrow \text{stack\_empty}) \vee (\neg\text{AlternatingColour}(col[a].\text{peek}(), col[b].\text{peek}()) \Rightarrow \text{not\_alternating\_colour}) \vee (\neg\text{DecreasingRank}(col[a$ not_decreasing_rank))

moveColToFree(a,b):

- transition: $col[a], fre[b] := col[a].remCard(), fre[b] = col[a].peek()$
- exception: $((\neg\text{ValidIndex}(8, 4, a, b) \Rightarrow \text{invalid\_index}) \vee (\text{StackEmpty}(col[a]) \Rightarrow \text{stack\_empty}) \vee (\neg\text{CellFree}(b) \Rightarrow \text{occupied\_cell}))$

moveFreeToCol(a,b):

- transition: $fre[a], col[b] := fre[a] = \text{CardT(NAS,NAR)}, col[a].\text{addCard}(fre[a])$

- exception: $((\neg\text{ValidIndex}(4, 8, a, b) \Rightarrow \text{invalid\_index}) \vee (\text{StackEmpty}(col[b]) \Rightarrow \text{stack\_empty}) \vee$ $(\text{CellFree}(a) \Rightarrow \text{occupied\_cell})) \vee (\neg\text{AlternatingColour}(fre[a], col[b].\text{peek}()) \Rightarrow \text{not\_alternating\_colour})$ $(\neg\text{DecreasingRank}(fre[a], col[b].\text{peek}()) \Rightarrow \text{not\_decreasing\_rank}))$

moveColToWin(a,b):

- transition: $col[a], fou[b] := col[a].\text{remCard}(), fou[b] = col[a].\text{peek}()$

- exception: $((\neg\text{ValidIndex}(8, 4, a, b) \Rightarrow \text{invalid\_index}) \vee (\text{StackEmpty}(col[a]) \Rightarrow \text{stack\_empty}) \vee$ $(\neg\text{SameSuit}(col[a].\text{peek}(), fou[b]) \Rightarrow \text{not\_same\_suit}) \vee (\neg\text{IncreasingRank}(fou[b], col[a].\text{peek}()) \Rightarrow$ $\text{not\_ascending\_rank})$

moveFreeToWin(a,b):

- transition: $fre[a], fou[b] := fre[a] = \text{CardT(NAS,NAR)}, fou[b] = col[a].\text{peek}()$

- exception: $((\neg\text{ValidIndex}(4, 4, a, b) \Rightarrow \text{invalid\_index}) \vee (\text{CellFree}(a) \Rightarrow \text{occupied\_cell})) \vee$ $(\neg\text{SameSuit}(fre[a], fou[b]) \Rightarrow \text{not\_same\_suit}) \vee (\neg\text{IncreasingRank}(fou[b], fre[a] \Rightarrow$ $\text{not\_ascending\_rank})$

isValidMoves():

- output $out := \exists(s : \text{StackT}|s \in col : \exists(c : \text{CardT}|c \in fou : \text{isIncreasingRank}(c, s.\text{peek}()) \wedge$ $\text{SameSuit}(c, s.\text{peek}()))) \vee \exists(c_1 : \text{CardT}|c_1 \in fre : \exists(c_2 : \text{CardT}|c_2 \in fou : \text{isIncreasingRank}(c_2, c_1) \wedge$ $\text{SameSuit}(c_1, c_2)))$
  $\vee\exists(s_1 : \text{StackT}|s_1 \in col : \exists(s_2 : \text{StackT}|s_2 \in col : s_1 \neq s_2 \wedge (isIncreasingRank(s_1.peek(), s_2.peek()) \vee$ $isDecreasingRank(s_1.peek(), s_2.peek())) \wedge iAlternatingRank(S_1.peek(), s_2.peek()) \wedge$ $\neg isStackEmpty(s_1) \wedge \neg isStackEmpty(s_2)))$
  $\vee\exists(c_1 : \text{CardT}|c_1 \in fre : \exists(s_1 : \text{StackT}|s_1 \in col : (\text{AlternatingColour}(c_1, s_1.\text{peek}) \wedge$ $(\text{IncreasingRank}(c_1, s_1.\text{peek}) \vee \text{DecreasingRank}(c_1, s_1.\text{peek})) \wedge c_1.\text{isValid}()) \vee (\neg c_1.\text{isValid}) \wedge$ $\neg\text{isStackEmpty}(s_1)))$

- exception: None

Game State 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	$x \in$		
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: $d := \forall s \in sprites : (getName(s) = "asteroid" \wedge getActivity(s) = false \wedge \exists c \in getChildren(s) : \neg isSafe(c) : false) \vee (getName(s) = "asteroid" \wedge getActivity(s) = true : checkCollision(obj, s, 50) : false) \vee (getName(s) \in \{"alien, "alienBullet"\} \wedge getActivity(s) = true \wedge checkCollision(obj, s, 50) : false)$

checkCollision(obj,other,r)

- input: $obj \in \text{GameObject}, other \in \text{GameObject}, obj \in$

togglePause():

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

Local Functions

screenShow: $String \times \{\text{NORMAL}, \text{EMPHASIS} \Rightarrow ?\}$ output: $out :=$
drawTriangle: $\mathbb{N} \Rightarrow$ output: $out :=$
getName: OBJECT $\Rightarrow$ output: $out :=$
getActivity: OBJECT $\Rightarrow$ output: $out :=$
getChildren: OBJECT $\Rightarrow$ output: $out :=$