SE 3XA3: Module Interface Specification

Team 10, MacLunky
Albert Zhou, zhouj103
Abeer Al-Yasiri, alyasira
Niyatha Rangarajan, rangaran

April 12, 2021

Table 1: Revision History

Date	**Developer(s)**	**Change**
March 16, 2021	Albert, Abeer, Niyatha	Version 0 made
March 31, 2021	Albert	Update to better match implementation
April 12, 2021	Albert, Niyatha	Srs updates based on consistency with other docs. Added state and input descriptions for all modules.

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

Entity Module

Template Module

Entity

Uses

None

Syntax

Exported Types

Entity = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Entity	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Entity	
move	$\mathbb{Z}$, $\mathbb{Z}$		
set	$\mathbb{N}$, $\mathbb{N}$		
position		[$\mathbb{N}$, $\mathbb{N}$]	
dimension		[$\mathbb{N}$, $\mathbb{N}$]	
damage	$\mathbb{Z}$, ReadMap		
overlap	Entity	$\mathbb{B}$	
tick	ReadMap, Mover		
useable		$\mathbb{B}$	
use	ReadMap, Mover		
flipImage			
makeEnt	string, $\mathbb{N}$, $\mathbb{N}$	Entity	

Semantics

State Variables

x: $\mathbb{N}$ - x coordinate
y: $\mathbb{N}$ - y coordinate
name: string - name
height: $\mathbb{N}$ - height
width: $\mathbb{N}$ - width

hp: $\mathbb{Z}$ - hit points
mat: image - display image

State Invariant

$x \geq 0 \wedge y \geq 0 \wedge height \geq 0 \wedge width \geq 0$

Assumptions

- Entity can only be instantiated by one of its subclasses.
- Images are surfaces for display.
- Images are a implementation decision.

Access Routine Semantics

new Entity$(x, y, name, height, width, hp, mat)$:

- output: Create an *Entity* object with the parameters $x, y, name, height, width, hp, mat$
- exception: None
- input definitions: x represents x-position, y represents y-position, $name$ represents what the Entity object is called, $height$ represents height dimension of Entity, $width$ represented width dimension of Entity, hp represents the health points of Entity, mat represents the loaded image of Entity.

move(a, b):

- transition: Change x, y by a, b
- exception: None
- input definition: a represents the distance an Entity object moves in the x-direction, b represents the distance an Entity object moved in the y-direction.

set(a, b):

- transition: Set x, y to a, b
- exception: None
- input definition: a represents the new Entity object x-position, b represents the new Entity object y-position.

position():

- output: x, y represents a tuple of the Entity object position.
- exception: None

dimension():

- output: $height, width$ represents a tuple of the Entity object size.
- exception: None

damage($d, gameinfo$):

- transition: Reduce hp by d
- exception: None
- input definition: d represents the damage value on the Entity, $gameinfo$ represents the state of the game Map.

overlap(ent):

- output: Whether or not ent, Entity object, overlaps with $self$ object.
- exception: None
- input definition: ent represents an Entity object that is near the $self$ object.

tick($gameinfo, player$):

- transition: None
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, $player$ represents the Mover object that is the user of the game controlling.

useable():

- output: $False$
- exception: None

use($gameinfo, player$):

- transition: None

- exception: None
- input definition: $gameinfo$ represents the state of the game Map, $player$ represents the Mover object that is the user of the game controlling.

flipImage():

- transition: Flip mat horizontally
- exception: None

makeEnt(n, x, y):

- output: Create a n-named object with parametersx, y
- exception: None
- input definition: n represents the name of the output Entity object, x represents the x-position of the Entity object, y represents the y-position of the Entity object.

Collectable Module

Template Module inherits Entity

Collectable

Uses

Entity

Syntax

Exported Types

Collectable = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Collectable	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Collectable	
useable		$\mathbb{B}$	
use	ReadMap, Mover		

Semantics

State Variables

val: $\mathbb{Z}$ - the amount of a resource contained in the object

- Collectable can only be instantiated by one of its subclasses.

Access Routine Semantics

new Collectable($x, y, name, height, width, hp, mat$):

- output: Create a *Collectable* object with the parameters $x, y, name, height, width, hp, mat$. Set val to 0.
- exception: None

- input definitions: x represents x-position, y represents y-position, *name* represents what the Collectable object is called, *height* represents height dimension of Collectable, *width* represented width dimension of Collectable, *hp* represents the health points of Collectable, *mat* represents the loaded image of Collectable.

useable():

- output: $True$
- exception: None

use($gameinfo, player$):

- transition: remove from $gameinfo$
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the user of the game controlling.

Treasure Module

Template Module inherits Collectable

Treasure

Uses

Collectable

Syntax

Exported Types

Treasure = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Treasure	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Treasure	
use	ReadMap, Mover		

Semantics

- Treasure can only be instantiated by one of its subclasses.

Access Routine Semantics

new Treasure($x, y, name, height, width, hp, mat$):

- output: Create a $Treasure$ object with the parameters $x, y, name, height, width, hp, mat$. Set val to 0.
- exception: None
- input definitions: x represents x-position, y represents y-position, $name$ represents what the Treasure object is called, $height$ represents height dimension of Treasure, $width$ represented width dimension of Treasure, hp represents the health points of Treasure, mat represents the loaded image of Treasure.

use($gameinfo, player$):

- transition: Use inherited use(). Add val to $player$ gold

- exception: None
- input definition: $gameinfo$ represents the state of the game Map, $player$ represents the Mover object that is the player of the game controls.

Gold Module

Template Module inherits Treasure

Gold

Uses

Treasure

Syntax

Exported Constants

VALGOLD = 500 - the amount of money a bar of gold is worth

Exported Types

Gold = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Gold	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Gold	

Semantics

Access Routine Semantics

new Gold($x, y, name, height, width, hp, mat$):

- output: Create a *Gold* object with the parameters $x, y, name, height, width, hp, mat$. Set *val* to VALGOLD.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Gold object is called, *height* represents height dimension of Gold, *width* represented width dimension of Gold, *hp* represents the health points of Gold, *mat* represents the loaded image of Gold.

Emerald Module

Template Module inherits Treasure

Emerald

Uses

Treasure

Syntax

Exported Constants

VALEMERALD = 800 - the amount of money an emerald is worth

Exported Types

Emerald = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Emerald	ℕ, ℕ, string, ℕ, ℕ, ℕ, image	Emerald	

Semantics

Access Routine Semantics

new Emerald($x, y, name, height, width, hp, mat$):

- output: Create an *Emerald* object with the parameters $x, y, name, height, width, hp, mat$. Set *val* to VALEMERALD.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Emerald object is called, *height* represents height dimension of Emerald, *width* represented width dimension of Emerald, *hp* represents the health points of Emerald, *mat* represents the loaded image of Emerald.

Sapphire Module

Template Module inherits Treasure

Sapphire

Uses

Treasure

Syntax

Exported Constants

VALSAPPHIRE = 500 - the amount of money a sapphire is worth

Exported Types

Sapphire = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Sapphire	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Sapphire	

Semantics

Access Routine Semantics

new Sapphire($x, y, name, height, width, hp, mat$):

- output: Create a *Sapphire* object with the parameters $x, y, name, height, width, hp, mat$. Set *val* to VALSAPPHIRE.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Sapphire object is called, *height* represents height dimension of Sapphire, *width* represented width dimension of Sapphire, *hp* represents the health points of Sapphire, *mat* represents the loaded image of Sapphire.

Ruby Module

Template Module inherits Treasure

Ruby

Uses

Treasure

Syntax

Exported Constants

VALRUBY = 1600 - the amount of money a ruby is worth

Exported Types

Ruby = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Ruby	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Ruby	

Semantics

Access Routine Semantics

new Ruby($x, y, name, height, width, hp, mat$):

- output: Create a *Ruby* object with the parameters $x, y, name, height, width, hp, mat$. Set *val* to VALRUBY.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Ruby object is called, *height* represents height dimension of Ruby, *width* represented width dimension of Ruby, *hp* represents the health points of Ruby, *mat* represents the loaded image of Ruby.

Diamond Module

Template Module inherits Treasure

Diamond

Uses

Treasure

Syntax

Exported Constants

VALDIAMOND = 5000 - the amount of money a diamond is worth

Exported Types

Diamond = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Diamond	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Diamond	

Semantics

Access Routine Semantics

new Diamond($x, y, name, height, width, hp, mat$):

- output: Create a *Diamond* object with the parameters $x, y, name, height, width, hp, mat$. Set *val* to VALDIAMOND.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Diamond object is called, *height* represents height dimension of Diamond, *width* represented width dimension of Diamond, *hp* represents the health points of Diamond, *mat* represents the loaded image of Diamond.

BombPile Module

Template Module inherits Collectable

BombPile

Uses

Collectable

Syntax

Exported Constants

VALBOMBPILE = 3 - the number of bombs in the object

Exported Types

BombPile = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new BombPile	ℕ, ℕ, string, ℕ, ℕ, ℕ, image	BombPile	
use	ReadMap, Mover		

Semantics

Access Routine Semantics

new BombPile($x, y, name, height, width, hp, mat$):

- output: Create a *BombPile* object with the parameters $x, y, name, height, width, hp, mat$. Set *val* to VALBOMBPILE.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the BombPile object is called, *height* represents height dimension of BombPile, *width* represented width dimension of BombPile, *hp* represents the health points of BombPile, *mat* represents the loaded image of BombPile.

use($gameinfo, player$):

- transition: Use inherited use(). Add *val* to *player* bombs
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

RopePile Module

Template Module inherits Collectable

RopePile

Uses

Collectable

Syntax

Exported Constants

VALROPEPILE = 3 - the number of ropes in the object

Exported Types

RopePile = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new RopePile	ℕ, ℕ, string, ℕ, ℕ, ℕ, image	RopePile	
use	ReadMap, Mover		

Semantics

Access Routine Semantics

new RopePile($x, y, name, height, width, hp, mat$):

- output: Create a *RopePile* object with the parameters $x, y, name, height, width, hp, mat$. Set *val* to VALROPEPILE.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the RopePile object is called, *height* represents height dimension of RopePile, *width* represented width dimension of RopePile, *hp* represents the health points of RopePile, *mat* represents the loaded image of RopePile.

use($gameinfo, player$):

- transition: Use inherited use(). Add *val* to *player* ropes
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

Sign Module

Template Module inherits Entity

Sign

Uses

Entity

Syntax

Exported Types

Sign = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Sign	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Sign	
setText	string		
getText		string	

Semantics

State Variables

text: string - the message on the Sign

Access Routine Semantics

new Sign($x, y, name, height, width, hp, mat, text$):

- output: Create a *Sign* object with the parameters $x, y, name, height, width, hp, mat$. Set *text* to empty string.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Sign object is called, *height* represents height dimension of Sign, *width* represented width dimension of Sign, *hp* represents the health points of Sign, *mat* represents the loaded image of Sign.

setText(t):

- transition: Set $text$ to t
- exception: None
- input definitions: t represent the text message to be displayed in the Sign object.

getText():

- output: $text$ represents the massage on the Sign object
- exception: None

Explosion Module

Template Module inherits Entity

Explosion

Uses

Entity

Syntax

Exported Constants

EXPLOSIONTIME = 45
- the duration of the explosion EXPLOSIONDAMAGE = 10 - the amount of damage dealt

Exported Types

Explosion = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Explosion	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Explosion	
tick	ReadMap, Mover		
explode	ReadMap, Mover		

Semantics

State Variables

time: $\mathbb{N}$ - the time remaining until the explosion is over

State Invariant

$0 \leq time \leq$ EXPLOSIONTIME

Access Routine Semantics

new Explosion($x, y, name, height, width, hp, mat$):

- output: Create an *Explosion* object with the parameters $x, y, name, height, width, hp, mat$. Set *time* to EXPLOSIONTIME.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Explosion object is called, *height* represents height dimension of Explosion, *width* represented width dimension of Explosion, *hp* represents the health points of Explosion, *mat* represents the loaded image of Explosion.

tick($gameinfo, player$):

- transition: Decrement *time*. If $time = 0$, remove from $gameinfo$
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

explode($gameinfo, player$):

- transition: Destroy nearby blocks. Deal EXPLOSIONDAMAGE damage to all nearby *Entity* objects and $Mover$ object
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

Rope Module

Template Module

Rope

Uses

Entity

Syntax

Exported Constants

LENGTH = 4 - the maximum travel distance

Exported Types

Rope = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Rope	ℕ, ℕ, string, ℕ, ℕ, ℕ, image	Rope	
tick	ReadMap, Mover		
throw			
makeRope	readMap		

Semantics

State Variables

ys: $\mathbb{Z}$ - movement speed in the y direction
$dist$: $\mathbb{N}$ - maximum distance the rope can travel

State Invariant

$ys \leq 0 \land 0 \leq dist \leq 32 * \text{LENGTH}$

Access Routine Semantics

new Rope($x, y, name, height, width, hp, mat$):

- output: Create a *Rope* object with the parameters $x, y, name, height, width, hp, mat$. Set $ys, dist$ to 0, 0.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Rope object is called, *height* represents height dimension of Rope, *width* represented width dimension of Rope, *hp* represents the health points of Rope, *mat* represents the loaded image of Rope.

tick($gameinfo, player$):

- : If $ys \neq 0$, move the object
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

throw():

- transition: Set ys to -2
- exception: None

makeRope($gameinfo$):

- transition: Change *mat* and *height* based on the rope length. The length is the minimum of the number of blocks until it reaches a solid block or LENGTH. Set x, y to the top middle of a block.
- exception: None
- input definition: $gameinfo$ represents the state of the game Map.

Local Functions

move: ReadMap $\times \mathbb{Z}$
move($gameinfo, ys$) $\equiv$
transition: Move along the y axis. If the object collides with a solid block or travels the maximum distance, set ys to 0 and make the rope.
input definition: $gameinfo$ represents the state of the game Map, ys represents the distance the rope will travel on the y-direction.

Throwable Module

Template Module

Throwable

Uses

Entity

Syntax

Exported Constants

GRAVITY = 3 - the speed of an object falling due to gravity
TOOFASTFORGRAVITY = 1 the speed of an object moving to be unaffected by gravity

Exported Types

Throwable = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Throwable	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Throwable	
tick	ReadMap, Mover		
pickUp			
putDown			
throw	$\mathbb{Z}$, $\mathbb{Z}$		
carry	$\mathbb{Z}$, $\mathbb{Z}$		
rem	ReadMap, Mover		
flip			

Semantics

State Variables

xs: $\mathbb{Z}$ - movement speed in the x direction
ys: $\mathbb{Z}$ - movement speed in the y direction
$held$: $\mathbb{B}$ - whether the object is held by the player

thrown: $\mathbb{B}$ - whether the object has been thrown
direction: $\mathbb{Z}$ - the direction the object is facing

Assumptions

- Throwable can only be instantiated by one of its subclasses.

Access Routine Semantics

new Throwable($x, y, name, height, width, hp, mat$):

- output: Create a *Throwable* object with the parameters $x, y, name, height, width, hp, mat$. Set $xs, ys, held, thrown, direction$ to 0, 0, False, False, 1.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Throwable object is called, *height* represents height dimension of Throwable, *width* represented width dimension of Throwable, *hp* represents the health points of Throwable, *mat* represents the loaded image of Throwable.

tick($gameinfo, player$):

- transition: If the object is not held, it is affected by gravity
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

pickUp():

- transition: Set *held* to True
- exception: None

putDown():

- transition: Set *held* to False
- exception: None

throw(x, y):

- transition: Set $xs, ys, held, thrown$ to x, y, False, True

- exception: None
- input definition: x represents the new x-position of object after being thrown, y represent the new y-position of the object after being thrown.

carry(a, b):

- transition: Change x, y by a, b
- exception: None
- input definition: a represents the x-distance moved of object when being carried, b represent the y-distance of the object when being carrried.

rem$(gameinfo, player)$:

- transition: Remove from $gameinfo$. Put it down. Remove it from $player$
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, $player$ represents the Mover object that is the player of the game controls.

flip():

- transition: Flip mat horizontally. Set $direction$ to $-direction$
- exception: None

Local Functions

gravity: ReadMap
gravity$(gameinfo) \equiv$
transition: If the magnitude of $xs \geq$ TOOFASTFORGRAVITY, set ys 0. Else, set ys to GRAVITY
input definition: $gameinfo$ represents the state of the game Map.
move: ReadMap $\times \mathbb{Z} \times \mathbb{Z}$
move$(gameinfo, xs, ys) \equiv$
transition: Move along the x and y axis. Set the appropriate xs or ys to 0 if the object collides with a solid block. input definition: $gameinfo$ represents the state of the game Map, xs represents distance of movement in the x-position, ys represents distance of movement in the y-position.

Chest Module

Template Module

Chest

Uses

Throwable

Syntax

Exported Types

Chest = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Chest	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Chest	
useable		$\mathbb{B}$	
use	ReadMap, Mover		

Semantics

Access Routine Semantics

new Chest($x, y, name, height, width, hp, mat$):

- output: Create a *Chest* object with the parameters $x, y, name, height, width, hp, mat$.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Chest object is called, *height* represents height dimension of Chest, *width* represented width dimension of Chest, *hp* represents the health points of Chest, *mat* represents the loaded image of Chest.

useable():

- $True$
- exception: None

use($gameinfo, player$):

- transition: Use inherited rem(). Create a random *Collectable* subsclass object with the same x, y. Add it to $gameinfo$
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

Bomb Module

Template Module

Bomb

Uses

Throwable, Explosion

Syntax

Exported Constants

BOMBTIME = 180 - the amount of time until the bomb explodes

Exported Types

Bomb = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Bomb	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Bomb	
tick	ReadMap, Mover		
explode	ReadMap, Mover		

Semantics

State Variables

time: $\mathbb{N}$ - the amount of time remaining until the bomb explodes

State Invariant

$0 \leq time \leq$ BOMTIME

Access Routine Semantics

new Bomb($x, y, name, height, width, hp, mat$):

- output: Create a *Bomb* object with the parameters $x, y, name, height, width, hp, mat$. Set *time* to BOMBTIME.

- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Bomb object is called, *height* represents height dimension of Bomb, *width* represented width dimension of Bomb, *hp* represents the health points of Bomb, *mat* represents the loaded image of Bomb.

tick($gameinfo, player$):

- transition: Decrement *time*. Use inherited tick(). If $time = 0$, the object is thrown into a solid block, or if it is destroyed, the object explodes
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

explode($gameinfo, player$):

- transition: Remove from $gameinfo$. Create an *Explosion* object with the same x, y. Add it to $gameinfo$. Explode it
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

EmptyHand Module

Template Module inherits Throwable

EmptyHand

Uses

Throwable

Syntax

Exported Types

EmptyHand = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new EmptyHand	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	EmptyHand	

Semantics

Access Routine Semantics

new EmptyHand($x, y, name, height, width, hp, mat$):

- output: Create a *EmptyHand* object with the parameters $x, y, name, height, width, hp, mat$.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the EmptyHand object is called, *height* represents height dimension of EmptyHand, *width* represented width dimension of EmptyHand, *hp* represents the health points of EmptyHand, *mat* represents the loaded image of EmptyHand.

Enemy Module

Template Module inherits Entity

Enemy

Uses

Entity

Syntax

Exported Constants

GRAVITY = 3 - the speed of an object falling due to gravity

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Enemy	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Enemy	
tick	ReadMap, Mover		
damage	$\mathbb{N}$, ReadMap	textcolorred	textcolorred

Semantics

State Variables

attack: $\mathbb{N}$ - the amount of damage dealt
xs: $\mathbb{Z}$ - movement speed in the x direction
ys: $\mathbb{Z}$ - movement speed in the y direction

Assumptions

- Enemy can only be instantiated by one of its subclasses.

Access Routine Semantics

new Enemy($x, y, name, height, width, hp, mat$):

- output: Create an *Enemy* object with the parameters $x, y, name, height, width, hp, mat$. Set $attack, xs, ys$ to 0, 0, 0.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Enemy object is called, *height* represents height dimension of Enemy, *width* represented width dimension of Enemy, *hp* represents the health points of Enemy, *mat* represents the loaded image of Enemy.

tick(gameinfo, player):

- transition: The object is affected by gravity. It moves. If it overlaps with *player*, deal *attack* damage to it
- exception: None
- input definition: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

damage($d, gameinfo$):

- transition: Use inherited damage(). If $hp \leq 0$, remove from $gameinfo$
- exception: None
- input definition: d represents the damage value on the affecting the Enemy object, $gameinfo$ represents the state of the game Map.

Local Functions

move: ReadMap × Mover
move($gameinfo$) ≡
transition: None.
input definition: $gameinfo$ represents the state of the game Map.

gravity:
gravity() ≡
transition: Set ys to GRAVITY

Snake Module

Template Module inherits Enemy

Snake

Uses

Enemy

Syntax

Exported Constants

SNAKEDAMAGE = 1 - the amount of damage a snake deals

Exported Types

Snake = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Snake	ℕ, ℕ, string, ℕ, ℕ, ℕ, image	Snake	

Semantics

Access Routine Semantics

new Snake($x, y, name, height, width, hp, mat$):

- output: Create a *Snake* object with the parameters $x, y, name, height, width, hp, mat$. Set $xs, attack$ to 0.5, SNAKEDAMAGE.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Snake object is called, *height* represents height dimension of Snake, *width* represented width dimension of Snake, *hp* represents the health points of Snake, *mat* represents the loaded image of Snake.

Local Functions

move: ReadMap
move($gameinfo$) $\equiv$
transition: Move along the x axis. If the object collides with a solid block or would fall off the edge of a solid block, turn it around. Move down the y axis. If the object collides with a solid block, set ys to 0.
input definition: $gameinfo$ represents the state of the game Map.

Spider Module

Template Module inherits Enemy

Spider

Uses

Enemy

Syntax

Exported Constants

SPIDEREDAMAGE = 1 - the amount of damage a spider deals
SPIDESENSE = 1 - the range a spider can detect the player

Exported Types

Spider = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Spider	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Snake	

Semantics

Access Routine Semantics

new Spider($x, y, name, height, width, hp, mat$):

- output: Create a *Spider* object with the parameters $x, y, name, height, width, hp, mat$. Set *attack* to SPIDERDAMAGE.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Spider object is called, *height* represents height dimension of Spider, *width* represented width dimension of Spider, *hp* represents the health points of Spider, *mat* represents the loaded image of Spider.

Local Functions

move: ReadMap × Mover
move($gameinfo, player$) ≡
transition: If *player* is not within SPIDERSENSE blocks, it doesn't move. Else, jump towards *player*
input definition: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

Trap Module

Template Module inherits Entity

Trap

Uses

Entity

Syntax

Exported Types

Trap = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Trap	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Trap	

Assumptions

- Trap can only be instantiated by one of its subclasses.

Access Routine Semantics

new Trap($x, y, name, height, width, hp, mat$):

- output: Create a $Trap$ object with the parameters $x, y, name, height, width, hp, mat$.
- exception: None
- input definitions: x represents x-position, y represents y-position, $name$ represents what the Trap object is called, $height$ represents height dimension of Trap, $width$ represented width dimension of Trap, hp represents the health points of Trap, mat represents the loaded image of Trap.

ArrowTrap Module

Template Module inherits Trap

ArrowTrap

Uses

Trap, Arrow

Syntax

Exported Constants

ARROWSENSE = 4 - the range an arrow trap can detect the player

Exported Types

ArrowTrap = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new ArrowTrap	$\mathbb{N}, \mathbb{N}, \mathbb{N}$,image,String,$\mathbb{N}, \mathbb{N}$	ArrowTrap	
tick	ReadMap, Mover		
setDirection	$\mathbb{Z}$		

Semantics

State Variables

$arrowMade$: $\mathbb{B}$ - whether or not the object has shot an arrow
$direction$: $\mathbb{Z}$ - the direction the object is facing

Access Routine Semantics

new ArrowTrap($x, y, name, height, width, hp, mat$):

- output: Create a $ArrowTrap$ object with the parameters $x, y, name, height, width, hp, mat$. Set $arrowMade$ to False.
- exception: None

- input definitions: x represents x-position, y represents y-position, *name* represents what the ArrowTrap object is called, *height* represents height dimension of ArrowTrap, *width* represented width dimension of ArrowTrap, *hp* represents the health points of ArrowTrap, *mat* represents the loaded image of ArrowTrap.

tick($gameinfo, player$):

- transition: If *player* is ARROWSENSE blocks in front of the object and $\neg arrowMade$, create an *Arrow* object, throw it towards *player*, add it to $gameinfo$, set $arrowMade$ to True
- exception: None
- input definitions: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

setDirection(d):

- transition: Set *direction* to d. If $direction = 1$, flip *mat* horizontally
- exception: None
- input definitions: d represents the direction of the object to be set to.

Arrow Module

Template Module inherits Throwable

Arrow

Uses

Throwable

Syntax

Exported Constants

ARROWDAMAGE = 2 - the amount of damage an arrow deals

Exported Types

Arrow = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Arrow	$\mathbb{N}, \mathbb{N}, \mathbb{N}$,image,String,$\mathbb{N}, \mathbb{N}$	Arrow	
pickup			
putDown			
tick	ReadMap, Mover		red

Semantics

State Variables

playerOwned: $\mathbb{B}$ - whether or not the arrow is thrown by the player

Access Routine Semantics

new Arrow($x, y, name, height, width, hp, mat$):

- output: Create an *Arrow* object with the parameters $x, y, name, height, width, hp, mat$. Set *playerOWned* to False.
- exception: None

- input definitions: x represents x-position, y represents y-position, *name* represents what the Arrow object is called, *height* represents height dimension of Arrow, *width* represented width dimension of Arrow, *hp* represents the health points of Arrow, *mat* represents the loaded image of Arrow.

pickup():

- transition: Use inherited pickup(). Set *playerOwned* to True
- exception: None

putDown():

- transition: Use inherited putdown(). Set *playerOwned* to False
- exception: None

tick(gameinfo, player):

- transition: Use inherited tick(). If moving horizontally, deal ARROWDAMAGE damage to the first *Mover* or *Enemy* object the arrow collides with and stop. If *playerOwned*, it ignores *Mover* objects.
- exception: None
- input definitions: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

Spikes Module

Template Module inherits Trap

Spikes

Uses

Trap

Syntax

Exported Constants

SPIKEDAMAGE = 4 - the amount of damage spikes deals

Exported Types

Spike = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Spikes	$\mathbb{N}, \mathbb{N}, \mathbb{N}$,image,String,$\mathbb{N}, \mathbb{N}$	Spikes	
tick	ReadMap, Mover		

Semantics

Access Routine Semantics

new Spikes($x, y, name, height, width, hp, mat$):

- output: Create a *Spikes* object with the parameters $x, y, name, height, width, hp, mat$.
- exception: None
- input definitions: x represents x-position, y represents y-position, *name* represents what the Spikes object is called, *height* represents height dimension of Spikes, *width* represented width dimension of Spikes, *hp* represents the health points of Spikes, *mat* represents the loaded image of Spikes.

tick($gameinfo, player$):

- transition: If *player* falls on a spikes object, deal SPIKEDAMAGE damage to it
- exception: None
- input definitions: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

Weapon Module

Template Module

Weapon

Uses

Entity

Syntax

Exported Types

Weapon = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Weapon	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Weapon	
tick	ReadMap, Mover		
swing			
carry	$\mathbb{Z}$, $\mathbb{Z}$		
flip			

Semantics

State Variables

damage: $\mathbb{N}$ - the amount of damage the weapon deals
speed: $\mathbb{N}$ - the maximum amount of time until the weapon can be used again
activeFrames: $\mathbb{N}$ - the maximum amount of time for the weapon to stay out
time: $\mathbb{N}$ - the amount of time remaining until the weapon can be used again
active: $\mathbb{N}$ - the amount of time remaining for the weapon to stay out
direction: $\mathbb{Z}$ - the direction the object is facing

State Invariant

$0 \leq time \leq \text{SPEED} \wedge 0 \leq active \leq \text{ACTIVEFRAMES}$

Assumptions

- Weapon can only be instantiated by one of its subclasses.

Access Routine Semantics

new Weapon($x, y, name, height, width, hp, mat$):

- output: Create a $Weapon$ object with the parameters $x, y, name, height, width, hp, mat$. Set $damage, speed, activeFrames, time, active$ to 0, 0, 0, 0, 0.
- exception: None
- input definitions: x represents x-position, y represents y-position, $name$ represents what the Weapon object is called, $height$ represents height dimension of Weapon, $width$ represented width dimension of Weapon, hp represents the health points of Weapon, mat represents the loaded image of Weapon.

tick($gameinfo, player$):

- transition: If $time > 0$, decrement $time$. If $active > 0$, deal $damage$ to all overlapping $Enemy$ objects, decrement $active$. Else, remove from $gameinfo$ and $player$
- exception: None
- input definitions: $gameinfo$ represents the state of the game Map, $player$ represents the Mover object that is the player of the game controls.

swing($gameinfo$):

- transition: If $time = 0$, set $time, active$ to $speed, activeFrames$. Add to $gameinfo$.
- exception: None
- input definitions: $gameinfo$ represents the state of the game Map.

carry(a, b):

- transition: Use inherited move()
- exception: None
- input definitions: a represents the x-distance moved of object when being carried, b represent the y-distance of the object when being carrried.

flip():

- transition: Flip mat horizontally. Set $direction$ to $-direction$
- exception: None

Whip Module

Template Module

Whip

Uses

Weapon

Syntax

Exported Constants

DAMAGE = 4 - the amount of damage a whip deals
SPEED = 45 - the maximum amount of time until a whip can be used again
ACTIVEFRAMES = 45 - the maximum amount of time for a whip to stay out

Exported Types

Whip = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Whip	$\mathbb{N}$, $\mathbb{N}$, string, $\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$, image	Whip	

Semantics

Access Routine Semantics

new Whip($x, y, name, height, width, hp, mat$):

- output: Create a $Whip$ object with the parameters $x, y, name, height, width, hp, mat$. Set $damage, speed, activeFrames$ to DAMAGE, SPEED, ACTIVEFRAMES.
- exception: None
- input definitions: x represents x-position, y represents y-position, $name$ represents what the Whip object is called, $height$ represents height dimension of Whip, $width$ represented width dimension of Whip, hp represents the health points of Whip, mat represents the loaded image of Whip.

Mover Module

Template Module

Mover

Uses

Entity

Syntax

Exported Constants

INVINCIBILITYTIME = 270 - the maximum amount of time until the player can be damaged again
STOMPDAMAGE = 1 - the amount of damage the player does when stomping on an *Enemy* object
FALLDAMAGE = 1 - the amount of damage the player takes when falling to the ground too fast
GRAVITY = 3 - the speed of an object falling due to gravity
MAXJUMP = 24 - the maximum distance the player can reach by jumping
STARTINGHP = 4 - the number of hp the player starts with
STARTINGBOMBS = 4 - the number of bombs the player starts with
STARTINGROPES = 4 - the number of ropes the player starts with

Exported Access Programs

Routine name	In	Out	Exceptions
new Mover	$\mathbb{N}, \mathbb{N}, \mathbb{Z}, \mathbb{Z}$	Mover	
position		$[\mathbb{N}, \mathbb{N}]$	
damage	$\mathbb{N}$		
loop	ReadMap		
emptyHand			

Semantics

Environmental Variables

keyboard: Scanner(System.in) - reading inputs from the keyboard

State Variables

xs: $\mathbb{Z}$ - movement speed in the x direction
ys: $\mathbb{Z}$ - movement speed in the y direction
$invincibility$: $\mathbb{Z}$ - the amount of time remaining until the player can be damaged again
$jumpDist$: $\mathbb{Z}$ - the distance the player has jumped
x: $\mathbb{N}$ - x coordinate
y: $\mathbb{N}$ - y coordinate
hp: $\mathbb{Z}$ - hit points
$direction$: $\mathbb{Z}$ - the direction the player is facing
$gold$: $\mathbb{Z}$ - the amount of gold the player has
$image$: image - display image
$hand$: $Throwable$ - the object in the player's hand
$bombs$: $\mathbb{Z}$ - the number of bombs the player has
$ropes$: $\mathbb{Z}$ - the number of ropes the player has
$adjustCamera$: $\mathbb{Z}$ - the amount to adjust the camera

State Invariant

$jumpDist \leq MAXJUMP \wedge jumpSpeed \leq 2$

Assumptions

- Mover objects moves within the map boundaries and can't move across solid blocks.
- Mover constructor is called once only for each game instance before any other access routine is called for that object or any other object in the game.
- Assume that view/interface game instance is already initialized before calling Mover constructor.
- Assume the media content required for Mover is loaded in the form of list of FILES.

Access Routine Semantics

new Mover($x, y, direction, speed$):

- output: Create a $Mover$ object with the parameters $x, y, direction, speed$. Set the remaining state variables to their inital values.
- exception: None

- input definitions: x represents x-position, y represents y-position, *direction* represents the direction of object when initialized, *speed* represents speed of the Mover object when initialized.

position():

- output: x, y
- exception: None

damage(d):

- transition: If $invincibility \leq 0$, reduce hp by d, set $invincibility$ to INIVICIBILITYTIME
- exception: None
- input definitions: d represent the damage value affecting the object.

loop($gameinfo$):

- transition: The player is affected by gravity, interpret controls, move the player, decrement $invincibility$, increment $adjustCamera$ magnitude
- exception: None
- input definitions: $gameinfo$ represents the state of the game Map.

emptyHand():

- transition: Set $hand$ to an $EmptyHand$ object
- exception: None

Local Functions

gravity($gameinfo$):

- transition: Modify ys with relation to gravity, jumping, and climbing
- exception: None
- input definitions: $gameinfo$ represents the state of the game Map.

jump($gameinfo, maxjump$):

- transition: Set $jumpDist, ys$ to 0, -GRAVITY
- exception: None
- input definitions: $gameinfo$ represents the state of the game Map, $maxjump$ represents the maximum jumping distance.

move($gameinfo, x, y$):

- transition: Move along the x and y axis If the player collides with a solid block in a direction, set the appropriates xs or ys to 0. If the mover lands on an *Enemy* object, deal STOMPDAMAGE to it and bounce off. If the mover lands on a solid block with $ys \geq 6$, take FALLDAMAGE. Move *hand* with the player
- exception: None
- input definitions: $gameinfo$ represents the state of the game Map, x represents the new x-position of object after moving, y represents the new y-position of object after moving.

climb($gameinfo, y$):

- transition: Move along the y axis. If the player collides with a solid block or would no longer be holding a *Rope* object, set ys to 0. Move *hand* with the player
- exception: None
- input definitions: $gameinfo$ represents the state of the game Map, y represents the new y-position of object after climbing.

controls($key, gameinfo$):

- transition:

Condition	Action
$key = TAB$	Use nearby useable *Entity*
$key = LEFTSHIFT \wedge hand \neq emptyHand$	Throw *hand*
$key = LEFTSHIFT \wedge hand = emptyHand$	Swing $Whip$
$key = LEFTARROW$	Move left
$key = RIGHTARROW$	Move right
$key = UPARROW$	Move/climb up
$key = DOWNARROW$	Move/climb down
$key = SPACE$	Jump
$key = ESC$	Terminate system
$key = B \wedge bombs > 0$	Create *Bomb* at mover
$key = V \wedge ropes > 0$	Create *Rope* at mover

- exception: None
- input definitions: *key* represents the key press input from the keyboard by the player of the game, $gameinfo$ represents the state of the game Map.

Display Module

Module

Display

Uses

Entities, Mover, ReadMap

Syntax

Exported Types

Display = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Display			Display
sign	Readmap		
resourceDisp	Mover		
goldDisp	Mover		
gameover	Mover		

Semantics

Enviroment Variables

screen: It is the game screen that is manipulated by the following functions to alter display. This is update by a precise frame rate to depict various game objects on the game screen.

State Variables

Access Routine Semantics

new Display()

- output: Create a *Display* object
- exception: None

sign($gameinfo, player$):

- transition: Show overlapping *Sign* object messages at the bottom of the screen
- exception: None
- input definitions: $gameinfo$ represents the state of the game Map, *player* represents the Mover object that is the player of the game controls.

resrouceDisp(*player*):

- transition: Show *player* hp, bombs, and ropes at the top left of screen
- exception: None
- input definitions: *player* represents the Mover object that is the player of the game controls.

goldDisp(*player*)

- transition: Show *player* gold under *player* hp
- exception: None
- input definitions: *player* represents the Mover object that is the player of the game controls.

gameover(*player*):

- transition: Show a game over message on the screen
- exception: None
- input definitions: *player* represents the Mover object that is the player of the game controls.

ReadMap Module

Template Module

ReadMap

Uses

None

Syntax

Exported Types

ReadMap = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new ReadMap	string	ReadMap	
add	Entity		
rem	Entity		
entities		seq of Entity	
enemies		seq of Enemy	
solid	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	
destroyBlock	$\mathbb{N}$, $\mathbb{N}$		

Semantics

State Variables

$gameMap$: seq of (seq of $\{image, \mathbb{N}, \mathbb{N}, \mathbb{B}, \mathbb{B}, string\}$ (image, x, y, transparent, solid, name)) defined as Block - all blocks in the game
$entList$: seq of Entity - the *Entity* objects in the game
$enemyList$: seq of Enemy - the *Enemy* objects in the game

Assumptions

- Map files contain a list of Entity names and their locations, and 2D sequence of map blocks.

Access Routine Semantics

new ReadMap(f):

- output: Create a *ReadMap* object from the data in the map file f.
- exception: None
- input definitions: f represents the name of the map file to be read.

add(*ent*):

- transition: Add *ent* to *entList*. If *ent* is an *Enemy* object, add *ent* to *enemyList*
- exception: None
- input definitions: *ent* represent an Entity object to be added to the map for display.

rem(*ent*):

- transition: Remove *ent* from *entList*. If *ent* is an *Enemy* object, remove *ent* from *enemyList*
- exception: None
- input definitions: *ent* represent an Entity object to be removed from the map to no longer display.

entities():

- output: *entityList*
- exception: None

enemies():

- output: *enemyList*
- exception: None

solid(x, y):

- output: Whether or not the block at x, y is solid
- exception: None

- input definitions: x represents the x-position to be tested, y represents the y-position to be tested.

destroyBlock(x, y):

- transition: Set the block at x, y to $\neg$transparent and $\neg$solid
- exception: None
- input definitions: x represents the x-position to destroy a block from, y represents the y-position to destroy a block from.

Main Module

Module

Main

Uses

Entities, Mover, Display, ReadMap

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
main			

Semantics

State Variables

$gameinfo$: ReadMap - the game level
$newmover$: Mover - the player character
$display$: Display - the display

Access Routine Semantics

main():

- transition: Create $gameinfo, newmover, display$.
  While the game is running, loop $newmover$, display $gameinfo$, display the *Entity* objects, tick the *Entity* objects, show $display$
- exception: None