

SE 3XA3:
Module Interface Specification
CraftMaster

Group Number: 307

Group Name: 3 Craftsmen

Members:

Hongqing Cao 400053625

Sida Wang 400072157

Weidong Yang 400065354

Contents

1	Block Module(M1)	1
1.1	Module	1
1.2	Uses	1
1.3	Syntax	1
1.3.1	Exported Constants	1
1.3.2	Exported Types	1
1.3.3	Exported Access Programs	1
1.4	Semantics	1
1.4.1	State Variables	1
1.4.2	State Invariant	1
1.4.3	Assumptions	1
1.4.4	Access Routine Semantics	2
1.5	Local Functions	2
2	Button Module(M2)	3
2.1	Module	3
2.2	Uses	3
2.3	Syntax	3
2.3.1	Exported Constants	3
2.3.2	Exported Types	3
2.3.3	Exported Access Programs of Button	3
2.3.4	Exported Access Programs of On Off Button	3
2.4	Semantics	4
2.4.1	Environment Variables	4
2.4.2	State Variables of Button	4
2.4.3	State Variables of On Off Button	4
2.4.4	State Invariant	4
2.4.5	Assumptions	4
2.4.6	Access Routine Semantics of Button	4
2.4.7	Access Routine Semantics of On Off Button	6
2.5	Local Functions	7
2.5.1	Local Functions of Button	7
2.5.2	Local Functions of On Off Button	7
3	Creature Module(M3)	8
3.1	Module	8
3.2	Uses	8
3.3	Syntax	8
3.3.1	Exported Constants	8
3.3.2	Exported Types	8
3.3.3	Exported Access Programs	8
3.4	Semantics	8
3.4.1	State Variables	8
3.4.2	State Invariant	9

3.4.3	Assumptions	9
3.4.4	Access Routine Semantics	9
4	Develop Tools Module(M4)	12
4.1	Module	12
4.2	Uses	12
4.3	Syntax	12
4.3.1	Exported Constants	12
4.3.2	Exported Types	12
4.3.3	Exported Access Programs	12
4.4	Semantics	12
4.4.1	State Variables	12
4.4.2	State Invariant	12
4.4.3	Assumptions	12
4.4.4	Access Routine Semantics	13
5	Game Module(M5)	14
5.1	Module	14
5.2	Uses	14
5.3	Syntax	14
5.3.1	Exported Constants	14
5.3.2	Exported Types	14
5.3.3	Exported Access Programs	14
5.4	Semantics	15
5.4.1	State Variables	15
5.4.2	State Variables	15
5.4.3	State Invariant	15
5.4.4	Assumptions	15
5.4.5	Access Routine Semantics	15
6	Load Source Module(M6)	19
6.1	Module	19
6.2	Uses	19
6.3	Syntax	19
6.3.1	Exported Constants	19
6.3.2	Exported Types	19
6.3.3	Exported Access Programs	19
6.4	Semantics	19
6.4.1	Environment Variables	19
7	Main Module(M7)	20
7.1	Module	20
7.2	Uses	20
7.3	Syntax	20
7.3.1	Exported Constants	20
7.3.2	Exported Types	20

7.3.3	Exported Access Programs	20
7.4	Semantics	20
7.4.1	State Variables	20
7.4.2	State Invariant	20
7.4.3	Assumptions	20
7.4.4	Access Routine Semantics	20
8	Game Scene Module(M8)	22
8.1	Module	22
8.2	Uses	22
8.3	Syntax	22
8.3.1	Exported Constants	22
8.3.2	Exported Types	22
8.3.3	Exported Access Programs	22
8.4	Semantics	22
8.4.1	Environment Variables	22
8.4.2	State Variables	23
8.4.3	State Invariant	23
8.4.4	Assumptions	23
8.4.5	Access Routine Semantics	23
8.5	Local Functions	25
9	Main Scene Module(M9)	26
9.1	Module	26
9.2	Uses	26
9.3	Syntax	26
9.3.1	Exported Constants	26
9.3.2	Exported Types	26
9.3.3	Exported Access Programs	26
9.4	Semantics	26
9.4.1	Environment Variables	26
9.4.2	State Variables	26
9.4.3	State Invariant	27
9.4.4	Assumptions	27
9.4.5	Access Routine Semantics	27
10	Setting Scene Module(10)	29
10.1	Module	29
10.2	Uses	29
10.3	Syntax	29
10.3.1	Exported Constants	29
10.3.2	Exported Types	29
10.3.3	Exported Access Programs	29
10.4	Semantics	29
10.4.1	Environment Variables	29
10.4.2	State Variables	29

10.4.3	State Invariant	30
10.4.4	Assumptions	30
10.4.5	Access Routine Semantics	30
11	Player Module(M11)	31
11.1	Module	31
11.2	Uses	31
11.3	Syntax	31
11.3.1	Exported Constants	31
11.3.2	Exported Types	31
11.3.3	Exported Access Programs	31
11.4	Semantics	31
11.4.1	State Variables	31
11.4.2	State Invariant	31
11.4.3	Assumptions	31
11.4.4	Access Routine Semantics	32
12	Process Queue Module(M12)	33
12.1	Module	33
12.2	Uses	33
12.3	Syntax	33
12.3.1	Exported Constants	33
12.3.2	Exported Types	33
12.3.3	Exported Access Programs	33
12.4	Semantics	33
12.4.1	State Variables	33
12.4.2	State Invariant	33
12.4.3	Assumptions	33
12.4.4	Access Routine Semantics	34
13	Screen Module(M13)	35
13.1	Module	35
13.2	Uses	35
13.3	Syntax	35
13.3.1	Exported Constants	35
13.3.2	Exported Types	35
13.3.3	Exported Access Programs	35
13.4	Semantics	35
13.4.1	Environment Variables	35
13.4.2	State Variables	35
13.4.3	State Invariant	36
13.4.4	Assumptions	36
13.4.5	Access Routine Semantics	36
13.5	Local Functions	37

14 Shape Module(M14)	39
14.1 Module	39
14.2 Uses	39
14.3 Syntax	39
14.3.1 Exported Constants	39
14.3.2 Exported Types	39
14.3.3 Exported Access Programs	39
14.4 Semantics	39
14.4.1 State Variables	39
14.4.2 State Invariant	39
14.4.3 Assumptions	39
14.4.4 Access Routine Semantics	40
15 World Module(M15)	41
15.1 Module	41
15.2 Uses	41
15.3 Syntax	41
15.3.1 Exported Constants	41
15.3.2 Exported Types	41
15.3.3 Exported Access Programs	41
15.4 Semantics	42
15.4.1 State Variables	42
15.4.2 State Invariant	42
15.4.3 Assumptions	42
15.4.4 Access Routine Semantics	42
15.5 Local Functions	45

List of Tables

1	Revision History	6
---	------------------	---

List of Figures

Date	Change
Mar 7	General Content added
Mar 8	Process Queue Module added
Mar 9	Load Source, Player, Shape, Develop Tools Module added
Mar 10	Block, Screen, Setting Scene, Main Scene, Game Scene Module added
Mar 11	Creature, Button, On Off Button, Game Module added
Mar 12	World Module added
Mar 13	Introduction added
Mar 13	Final edit

Table 1: **Revision History**

Introduction

This document acts as a blackbox description of CraftMaster modules. The potential readers should be able to figure out the modules' observable behaviors(Interfaces). The internal design is not accessible in this document. The modules specified in this document are labeled as **M?** and those labels are related to the module labels in the **Module Guide(MG)** for the seek of consistency.

This document support for the four key roles in software development as follows:

- **Designer:** A medium for design and review.
- **Developer:** A clear statement of the required task.
- **Tester:** A metric for correctness.
- **User:** Freedom of using modules from having to know internal details of them.

1 Block Module(M1)

1.1 Module

Block

1.2 Uses

[pygame.graphics.TextureGroup](#)

[pygame.image](#)

1.3 Syntax

1.3.1 Exported Constants

None

1.3.2 Exported Types

Block = ?

1.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
Block	str, tuple of float*3, float, str, bool		

1.4 Semantics

1.4.1 State Variables

Name: str

coordinates: list of float

texture: loaded texture file

destroyable: bool

1.4.2 State Invariant

None

1.4.3 Assumptions

Block() is called before any other access routine

1.4.4 Access Routine Semantics

Block(name, top, bottom, side, size, texturePath,destroyable = True):

- transition: name := name
coordinates := TextureGroup(image.load(texturePath).get_texture())
destroyable := destroyable
- exception: None

1.5 Local Functions

_tex_coords(top, bottom, side, size):

- transition: top := _tex_coord(*top,size)
bottom := _tex_coord(*bottom, size)
side := _tex_coord(*side, size)
result := [*top, *bottom, *(size×4)]
Note: *var is a sequence of variables of arbitrary types. e.g. $*(x,y) = x,y$
- out: *out* := *result*
- exception: None

_tex_coord(x, y, n):

- transition: $m := 1.0/n$
 $dx := x \times m$
 $dy := y \times m$
- out: *out* := $dx, dy, dx + m, dy, dx + m, dy + m, dx, dy + m$
- exception: None

2 Button Module(M2)

2.1 Module

Button
On Off Button

2.2 Uses

shape.Shape2D
[pygame.gl](#)
[pygame.graphics.TextureGroup](#)

2.3 Syntax

2.3.1 Exported Constants

None

2.3.2 Exported Types

Buttton = ?
On Off Button = ?

2.3.3 Exported Access Programs of Button

Routine name	In	Out	Exceptions
Button	float, float, int, int, str, tuple of int, tuple of int		
draw			
on_click	float, float	bool	
on_mouse	float, float, tuple of int, tuple of int		ValueError
on_resize	float, float, int, int		
changeFunc	list of tuple		
changeText	str		

2.3.4 Exported Access Programs of On Off Button

Routine name	In	Out	Exceptions
OnOffButton	float×2, int×2, list of tuple×2, tuple of int×3, bool, str×2		
draw			
on_click	float, float	bool	
on_resize	float, float, int, int		
changeState	bool		

2.4 Semantics

2.4.1 Environment Variables

Mouse

2.4.2 State Variables of Button

x, y: float, float
width, height: int, int
textColor: (int, int, int, int)
quadColor: (int, int, int)
funcList: [(func, str)]
text: str
label: label
quad: vertex_list

2.4.3 State Variables of On Off Button

x, y: float, float
width, height: int, int
state: bool
LeftToRightFunc, RightToLeftFunc: [(func, str)], [(func, str)]
leftText, rightText: label, label
quad, slideQuad: vertex_list, vertex_list

2.4.4 State Invariant

None

2.4.5 Assumptions

Button() and OnOffButton() is called before any other access routine

2.4.6 Access Routine Semantics of Button

Button(x, y, width, height, text, textColor, quadColor):

- transition: x, y := x, y
width, height := width, height
textColor, quadColor := textColor, quadColor
funcList := Null
text := text
label := Label(text, 'Arial', 3 × height//8, x + width//2, y + height//2, 'center', 'center', textColor)
quad := vertex_list(4, ('v2i', Shape2D.quad_vertices(x, y, width, height)), ('c3B', quadColor × 4))
- exception: None

draw():

- transition: present the button and label
- exception: None

on_click(x, y):

- transition: $_checkMouse(x,y) \Rightarrow ((func, args := function, func(*args))$ for function in funcList)
- output:

	<i>out :=</i>
$_checkMouse(x,y)$	True
$\neg _checkMouse(x,y)$	False

- exception: None

on_mouse(x, y, textColor, quadColor):

- transition:

$_checkMouse(x,y)$	label.color := textColor quad.colors := quadColor*4
$\neg _checkMouse(x,y)$	label.color := self.textColor quad.colors := self.quadColor*4

- exception:

	<i>exc :=</i>
$\text{len}(\text{textColor}) \neq 4 \vee \exists i \in \text{textColor} \cdot (i < 0 \vee i > 255)$	ValueError
$\text{len}(\text{quadColor}) \neq 3 \vee \exists i \in \text{quadColor} \cdot (i < 0 \vee i > 255)$	ValueError

on_resize(x, y, width, height):

- transition: $x, y := x, y$
width, height := width, height
label.x := $x + \text{width} // 2$
label.y := $y + \text{height} // 2$
label.font_size := $3 * \text{height} / 8$
quad.vertices := Shape2D.quad_vertices(x, y, width, height)
- exception: None

changeFunc(funcList):

- transition: funcList := funcList
- exception: None

2.4.7 Access Routine Semantics of On Off Button

OnOffButton(x, y, width, height, LeftToRightFunc, RightToLeftFunc, text-Color = (255,0,0,255), quadColor = (0,0,0), slideQuadColor = (64,64,64), state = False, leftText = "OFF", rightText = "ON"):

- transition: x, y := x, y
width, height := width, height
state := state
LeftToRightFunc, RightToLeftFunc := LeftToRightFunc, RightToLeftFunc
leftText := Label(leftText, 'Arial', 3 × height//7, x + 10 + width//2 - 10, y, 'right', 'center', textColor)
rightText := Label(rightText, 'Arial', 3 × height//7, x - width//2 - 10, y, 'left', 'center', textColor)
quad := vertex_list(4, ('v2i', Shape2D.quad_vertices(x - width//2, y - height//2, width, height)), ('c3B', quadColor × 4))
slideQuad := vertex_list(4, ('v2i', Shape2D.quad_vertices(x - 3 × width//8, y - 5 × height//2, width//4, 5 × height//4)), ('c3B', slideQuadColor × 4))
- exception: None

draw():

- transition: present the button and label
- exception: None

on_resize(x, y, width, height):

- transition: x, y := x, y
width, height := width, height
label.x := x + width//2
label.y := y + height//2
label.font_size := 3 * height/8
quad.vertices := Shape2D.quad_vertices(x, y, width, height)
- exception: None

on_click(x, y):

- transition:

$_checkMouseOn(x,y) \wedge state$	funcList := RightToLeftFunc
$_checkMouseOn(x,y) \wedge \neg state$	funcList := LeftToRightFunc

(func, args := function, func(*args)) for function in funcList

- output:

	<i>out</i> :=
<code>_checkMouseOn(x,y)</code>	True
<code>¬_checkMouseOn(x,y)</code>	False

- exception: None

`changeState(state)`:

- transition:

<code>state</code>	<code>slideQuad.vertices := Shape2D.quad_vertices(x + width//8, ...)</code>
<code>¬ state</code>	<code>slideQuad.vertices := Shape2D.quad_vertices(x - 3 × width//8, ...)</code>

Note: same parameters for both case are omitted to fit in table
`state := state`

- exception: None

2.5 Local Functions

2.5.1 Local Functions of Button

`_checkMouse(x, y)`:

- output: $out := x > x \wedge x < x + width \wedge y > y \wedge y < y + height$
- exception: None

2.5.2 Local Functions of On Off Button

`_checkMouseOn(x, y)`:

- output: $out := x - width//2 < x \wedge x < x + width//2 \wedge y - width//2 < y \wedge y < y + height//2$
- exception: None

3 Creature Module(M3)

3.1 Module

Creature

3.2 Uses

[math](#)

3.3 Syntax

3.3.1 Exported Constants

None

3.3.2 Exported Types

Creature = ?

3.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
Creature	tuple of float, int, float, float, bool, float, float, float		
rotate	int, int		
move	str		ValueError
stopMove	str		ValueError
jump	float		
update	float, world		
get_motion_vector		tuple of float	

3.4 Semantics

3.4.1 State Variables

position: tuple of float

rotation: tuple of float

strafe: list of int

dy: float

walkSpeed: float

JumpHeight: float

floating: bool

flySpeed: float

height: float

health: int

3.4.2 State Invariant

None

3.4.3 Assumptions

Creature() is called before any other access routine

3.4.4 Access Routine Semantics

Creature(position, health, dy = 0, walkSpeed = 5, flying = False, flySpeed = 10, height = 1, jumpHeight = 1.0):

- transition: position := position
rotation := (0, 0)
strafe := [0, 0]
dy := dy
walkSpeed := walkSpeed
JumpHeight := jumpHeight
terminalVelocity := 50
flying := flying
flySpeed := flySpeed
height := height
health := health
- exception: None

rotate(x, y):

- transition: $y := \max(-90, \min(90, y))$
rotation := (x, y)
- exception: None

move(direction):

- transition:

direction = "FORWARD"	strafe[0] := strafe[0] - 1
direction = "BACKWARD"	strafe[0] := strafe[0] + 1
direction = "LEFT"	strafe[1] := strafe[1] - 1
direction = "RIGHT"	strafe[1] := strafe[1] + 1

- exception: $exc := ((\text{direction} \neq \text{"FORWARD"} \wedge \text{direction} \neq \text{"BACKWARD"} \wedge \text{direction} \neq \text{"LEFT"} \wedge \text{direction} \neq \text{"RIGHT"}) \Rightarrow \text{ValueError}$

stopMove(x, y, dx, dy):

- transition:

direction = "FORWARD"	strafe[0] := strafe[0] + 1
direction = "BACKWARD"	strafe[0] := strafe[0] - 1
direction = "LEFT"	strafe[1] := strafe[1] + 1
direction = "RIGHT"	strafe[1] := strafe[1] - 1

- exception: $exc := ((direction \neq \text{"FORWARD"} \wedge direction \neq \text{"BACKWARD"} \wedge direction \neq \text{"LEFT"} \wedge direction \neq \text{"RIGHT"}) \Rightarrow \text{ValueError}$

jump(gravity):

- transition: $dy := \sqrt{2 \times gravity \times \text{JumpHeight}}$
- exception: None

update(dt, world):

- transition: speed := walkSpeed
 $d := dt \times speed$
 $dx, dy, dz := .get_motion_vector()$
 $dx, dy, dz := dx \times d, dy \times d, dz \times d$
Note: transitions above are for walking

flying	$dy := dy - dt \times gravity$ $dy := \max(dy, -terminalVelocity)$ $dy := dy + dy \times dt$
--------	--

Note: transitions above are for flying

$x, y, z := position$
 $x, y, z := world.collide((x + dx, y + dy, z + dz), self)$
 $position := (x, y, z)$
Note: transitions above are for collision

- exception: None

get_motion_vector():

- transition:

\exists strafe			x, y := rotation strafe := arctan(*strafe) y_angle := y x_angle := (x + strafe)
	flying		m := cos(y_angle) dy := sin(y_angle)
		strafe[1] = 0	dy := 0.0 m = 1
		strafe[1] > 0	dy := dy \times -1 dx := cos(x_angle) \times m dz := sin(x_angle) \times m
	\neg flying		dy := 0.0 dx := cos(x_angle) dz := sin(x_angle)
$\neg \exists$ strafe			dy := 0.0 dx := 0.0 dz := 0.0

- output: $out := (dx, dy, dz)$
- exception: None

4 Develop Tools Module(M4)

4.1 Module

None

4.2 Uses

[PIL.Image](#)

[os](#)

[math](#)

4.3 Syntax

4.3.1 Exported Constants

None

4.3.2 Exported Types

None

4.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
MergePicture	float, float, list of str, str, str	Null	ValueError

Note: “Null” value is represented as “None” in Python. In case of confusion with “None” used in other part of this document, we decided to use “Null” to represent this value.

4.4 Semantics

4.4.1 State Variables

None

4.4.2 State Invariant

None

4.4.3 Assumptions

None

4.4.4 Access Routine Semantics

MergePicture(width,height,pictures,folder, savePath):

- transition: single image := a join of multiple images
- out:

$len(pictures) = 0$	$out :=$
	Null

- exception: $exc := (type(savePath) \neq str \vee savePath = "" \Rightarrow \text{ValueError})$

5 Game Module(M5)

5.1 Module

Game

5.2 Uses

[__future__.division](#)
[json](#)
[os](#)
[collections.deque](#)
[pyglet.image](#) [pyglet.gl](#)
[pyglet.graphics.TextureGroup](#)
[pyglet.window](#)
[pyglet.window.key](#)
[player.Player](#)
[world.World](#)
[gameScene.GameScene](#)
[mainScene.MainScene](#)
[settingScene.SettingScene](#)
[loadSource.ALLBLOCKS](#)
[loadSource.PLACEBLOCKS](#)

5.3 Syntax

5.3.1 Exported Constants

None

5.3.2 Exported Types

Game = ?

5.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
Game	int, *args, **kwargs		
loadGame	str		ValueError
saveGame	str		
goBack			
StartNewGame			
changeScene	Screen	Null	ValueError
update	float		
on_mouse_press	float, float, button, int		
on_mouse_motion	float, float, float, float		
on_key_press	symbol, int		
on_key_release	symbol, int		
on_resize	int, int		
on_draw			

5.4 Semantics

5.4.1 State Variables

Mouse
 Keyboard
 Monitor
 Saving files

5.4.2 State Variables

player: Creature
 width, height: int, int
 world: World
 refreshRate: int
 setScene: Screen
 gameScene: Screen
 mainScene: Screen
 lastScene: Screen
 currentScene: Screen

5.4.3 State Invariant

None

5.4.4 Assumptions

Game() is called before any other access routine

5.4.5 Access Routine Semantics

Game(refreshRate = 60, *args, **kwargs):

- transition: `player := Player((0,0,0))`
`world := World(ALLBLOCKS)`
`refreshRate := refreshRate`
`setScene := SettingScene()`
`gameScene := GameScene(PLACEBLOCKS)`
`mainScene := MainScene()`
`lastScene, currentScene := Null`
- exception: `None`

`loadGame(file):`

- transition: `f := open(file, 'r')`
`data := json.load(f)`
`player.position := data["position"]`
`world := {block in data["world"]: (pos in data["world"][block])}`
`world.changeWorld(world)`
- exception: `exc := ¬ os.path.exists(file) ⇒ ValueError`

`saveGame(file):`

- transition: `player.position := data["position"]`
`data := {block in ALLBLOCKS: (pos in world["world"][block]), "position": player.position, "world": world}`
`world.changeWorld(world)` `f := open(file, 'w')`
`json.dump(data, f)`
- exception: `None`

`goBack():`

- transition: `currentScene := lastScene`
`set_exclusive_mouse(currentScene.mouseExclusive)`
- exception: `None`

`StartNewGame():`

- transition: `position := (0,0,0)`
`world.clearWorld()`
`world.setupWorld()`
`changeScene("game")`
- exception: `None`

changeScene(scene):

- transition: lastScene := currentScene

scene = "main"	currentScene := mainScene
scene = "game"	currentScene := gameScene
scene = "set"	currentScene := setScene

- output: $out := (scene = currentScene) \Rightarrow \text{Null}$
- exception: $exc := (scene \neq \text{"main"} \wedge scene \neq \text{"game"} \wedge scene \neq \text{"set"}) \Rightarrow \text{ValueError}$

update(dt):

- transition: currentScene.update(dt)
- exception: None

on_mouse_press(x, y, button, modifiers):

- transition: currentScene.mouseClick(x, y, button, modifiers)
- exception: None

on_mouse_motion(x, y, dx, dy):

- transition: currentScene.mouseMove(x, y, dx, dy)
- exception: None

on_key_press(symbol, modifiers):

- transition: currentScene.keyPressed(symbol, modifiers)
- exception: None

on_key_release(symbol, modifiers):

- transition: currentScene.keyRelease(symbol, modifiers)
- exception: None

on_resize(width, height):

- transition: `mainScene.screenResize(width,height)`
`gameScene.screenResize(width,height)` `setScene.screenResize(width,height)`
- exception: None

`on_draw()`:

- transition: `clear()`
`currentScene.draw()`
- exception: None

6 Load Source Module(M6)

6.1 Module

loadSource

6.2 Uses

[os](#)

[math](#)

[block.Block](#)

[pyglet.gl](#)

6.3 Syntax

6.3.1 Exported Constants

```
ICON = media.load("source\icon.png")
BUILDSOUND = media.load("source\build.wav", streaming=False)
DESTROYSOUND = media.load("source\destroy.wav", streaming=False)
BACKGROUNDMUSIC = media.load("source\bgmusic.wav")
```

```
ALLBLOCKS = [BRICK, GRASS, STONE, MARBLE]
```

```
PLACEBLOCKS = [BRICK, GRASS, STONE]
```

6.3.2 Exported Types

```
BRICK = Block("BRICK", (0, 0), (0, 0), (0, 0), 1, "texture\Brick.png", True)
GRASS = Block("GRASS", (0, 0), (0, 1), (1, 1), 2, "texture\Grass.png", True)
STONE = Block("STONE", (0, 0), (0, 0), (0, 0), 1, "texture\Stone.png", True)
MARBLE = Block("MARBLE", (0, 0), (0, 0), (0, 0), 1, "texture\Marble.png",
False)
```

6.3.3 Exported Access Programs

None

6.4 Semantics

6.4.1 Environment Variables

Source files

Texture files

7 Main Module(M7)

7.1 Module

7.2 Uses

pyglet.gl

game.Game

loadSource

7.3 Syntax

7.3.1 Exported Constants

WINDOW_WIDTH = 800

WINDOW_HEIGHT = 600

7.3.2 Exported Types

None

7.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
main			

7.4 Semantics

7.4.1 State Variables

None

7.4.2 State Invariant

None

7.4.3 Assumptions

None

7.4.4 Access Routine Semantics

Main():

- transition: `game := Game(WINDOW_WIDTH, WINDOW_HEIGHT, 'Craft-Man', True, 100)`
`game.set_icon(ICON)`

```
BACKGROUND MUSIC.play()  
pyglet.app.run()
```

- exception: None

8 Game Scene Module(M8)

8.1 Module

GameScene

8.2 Uses

sys
time
pyglet.window
pyglet.window.key
screen.Screen
loadSource
shape.Shape3D

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Types

GameScene = Screen

8.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
GameScene	game, list of block		
draw			
mouseClick	int, int, button, int		
mouseMove	int, int, tuple of int, tuple of int		
keyPressed	symbol, int		
keyRelease	symbol, int		
screenResize	int, int		
update	float		

8.4 Semantics

8.4.1 Environment Variables

Mouse
Keyboard
Monitor

8.4.2 State Variables

label: label
 reticle: ?
 inventory: list of str
 block: str
 num_keys: list of symbol

8.4.3 State Invariant

None

8.4.4 Assumptions

GameScene(game) is called before any other access routine

8.4.5 Access Routine Semantics

MainScene(game, placeBlocks):

- transition: mouseExclusive := True
 label := Label(',', font_name='Arial', font_size=18, x=10, y=height - 10, anchor_x='left', anchor_y='top', color=(255,255,255, 255))
 reticle := Null
 inventory := [block.name for block in placeBlocks]
 num_keys := [key..1, key..2, key..3, key..4, key..5, key..6, key..7, key..8, key..9, key..0]
- exception: None

mouseClick(x, y, button, modifiers):

- transition:

mouseExclusive	vector := get_sight_vector() curPos, prePos := hit_test(position, vector)
\neg mouseExclusive	mouseExclusive := True

mouseExclusive	
button = mouse.RIGHT \wedge prePos	add_block(prePos, block) BUILDSOUND.play()
button = mouse.LEFT \wedge curPos	block := world[curPos]

mouseExclusive \wedge button = mouse.LEFT \wedge curPos	
block \in inventory	remove_block(curPos) DESTROYSOUND.play()

- exception: None

mouseMove(x, y, dx, dy):

- transition:

mouseExclusive	$m := 0.15$ $x, y := \text{rotation}$, $x, y := x + dx \times m, y + dy \times m$ $\text{rotate}(x,y)$
----------------	--

- exception: None

keyPressed(symbol, modifiers):

- transition:

symbol = key.W	player.move("FORWARD")
symbol = key.S	player.move("BACKWARD")
symbol = key.A	player.move("LEFT")
symbol = key.D	player.move("RIGHT")
symbol = key.SPACE ^ player.dy = 0	player.jump(gravity)
symbol = key.ESCAPE	changeScene('set')
symbol = key.TAB	player.switchFlyState()
symbol ∈ num_keys	$\text{index} := (\text{symbol} - \text{num_keys}[0]) \% \text{len}(\text{inventory})$ $\text{block} := \text{inventory}[\text{index}]$

- exception: None

screenResize(width, height):

- transition:
 - label.y := height - 10
 - x, y := resize(width//2, width//2)
 - n := 10
 - reticle.delete(), if there is a reticle
 - reticle := vertex_list(4, ('v2i', (x - n, y, x + n, y, x, y - n, x, y + n)))
- exception: None

update(dt):

- transition:
 - updateWorld(1.0/refreshRate, player)
 - m := 8
 - dt := min(dt, 0.2)
 - reticle.delete(), if there is a reticle
 - m × player.update(dt/m, self.game.world)

- exception: None

draw():

- transition: present background, fog, label, focused block, and reticle
- exception: None

8.5 Local Functions

_draw_label():

- transition: `x, y, z := player.position`
`label.text = 'get_fps(), x, y, z, len(world._shown), len(world)'`
- exception: None

_draw_reticle():

- transition: `glColor3d(0, 0, 0)` *Note: black*
- exception: None

_drawFocusedBlock():

- transition: `vector := player.get_sight_vector()`
`block := world.hit_test(player.position, vector)[0]`
`x, y, z := block`
`vertex_data = Shape3D.cube_vertices(x, y, z, 0.51)`
`graphics.draw(24, GL_QUADS, ('v3f/static', vertex_data))`
Note: Transitions above draw black edge lines around the block that player currently focus.
- exception: None

9 Main Scene Module(M9)

9.1 Module

MainScene

9.2 Uses

screen.Screen
button.Button

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Types

MainScene = Screen

9.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
MainScene	game		
draw			
mouseClick	int, int, button, int		
mouseMove	int, int, tuple of int, tuple of int		
screenResize	int, int		
changeStage	str		

9.4 Semantics

9.4.1 Environment Variables

Mouse
Monitor

9.4.2 State Variables

game: game
stage: {str: list of tuple} button1: Button
button2: Button
button3: Button
buttons: list of Button

9.4.3 State Invariant

None

9.4.4 Assumptions

MainScene(game) is called before any other access routine

9.4.5 Access Routine Semantics

MainScene(game):

- transition: mouseExclusive := False
 $x, y := \text{game.width}, \text{game.height}$
stage := {
 "stage1": [
 ("start Game", [(changeStage, ("stage2",))]),
 ("Setting", [(changeScene, ("set",))]),
 ("Quit", [(close, ())]),
],
 "stage2": [
 ("Start New Game", [(self.game.StartNewGame, ())]),
 ("Load Game", [(changeStage, ("stage3",))]),
 ("Return", [(changeStage, ("stage1",))]),
],
 "stage3": [
 ("Game one", [(loadGame, ("game1.json",)), (changeScene, ("game",))]),
 ("Game two", [(loadGame, ("game2.json",)), (changeScene, ("game",))]),
 ("Return", [(changeStage, ("stage2",))]),
],
}
 button1 := Button(x//4, y//2, x//2, y//8, "", (255,255,255,255), (0,0,0))
 button2 := Button(x//4, (11*y)//32, x//2, y//8, "", (255,255,255,255), (0,0,0))
 button3 := Button(x//4, (3*y)//16, x//2, y//8, "", (255,255,255,255), (0,0,0))
 buttons := [button1, button2, button3]

- exception: None

mouseClick(x, y, button, modifiers):

- transition: detect mouse click on each button
- exception: None

mouseMove(x, y, dx, dy):

- transition: detect mouse move in window

- exception: None

screenResize(width, height):

- transition: $x, y := \text{width}, \text{height}$
button1 := resize($x//4, y//2, x//2, y//8$)
button2 := resize($x//4, (11*y)//32, x//2, y//8$)
button3 := resize($x//4, (3*y)//16, x//2, y//8$)
- exception: None

draw():

- transition: present background and buttons
- exception: None

changeStage(stage):

- transition: change presented for each stage
- exception: None

10 Setting Scene Module(10)

10.1 Module

SettingScene

10.2 Uses

screen.Screen
button.OnOffButton
button.Button

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Types

SettingScene = Screen

10.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
SettingScene	game		
screenResize	int, int		
draw			
mouseMove	int, int, tuple of int, tuple of int		
mouseClick	int, int, button, int		

10.4 Semantics

10.4.1 Environment Variables

Mouse
Monitor

10.4.2 State Variables

mode: OnOffButton
returnBut: Button
saveOneBut: Button
saveTwoBut: Button

10.4.3 State Invariant

None

10.4.4 Assumptions

SettingScene(game) is called before any other access routine

10.4.5 Access Routine Semantics

SettingScene(game):

- transition: mouseExclusive := False
 $x, y := \text{game.width}, \text{game.height}$
mode := OnOffButton("night", "day")
returnBut := Button("return")
saveOneBut := Button("Save on game 1 and Return")
saveTwoBut := Button("Save on game 2 and Return")
- exception: None

screenResize(width, height):

- transition: mode := resize(width//2, height//2, width//8, height//8)
returnBut := resize(20, height-20-width//24, width//8, width//20)
saveOneBut := resize(5*width//48, height//10, width//3, width//24)
saveTwoBut := resize(9*width//16, height//10, width//3, width//24)
Note: "//" here and below is integer division, that $5//2 = 2$
- exception: None

draw():

- transition: show background and all buttons
- exception: None

mouseMove(x, y, dx, dy):

- transition: detect mouse move in window
- exception: None

mouseClick(x, y, button, modifiers):

- transition: detect mouse click on each button
- exception: None

11 Player Module(M11)

11.1 Module

player

11.2 Uses

[math](#)

creature.Creature

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Types

Player = Creature

11.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
Player	position		
get_sight_vector		tuple of float	
switchFlyState			

11.4 Semantics

11.4.1 State Variables

None

11.4.2 State Invariant

None

11.4.3 Assumptions

Player() is called before any other access routine

11.4.4 Access Routine Semantics

Player(position):

- transition: $Player := (position, 2, 100)$
 $energy := 100$
- exception: none

get_sight_vector():

- transition: $x, y := \text{rotation}$
 $m := \cos(y)$ $dy := \sin(y)$
 $dx := \cos(x - 90) \times m$
 $dz := \sin(x - 90) \times m$
- output: $out := dx, dy, dz$
- exception: none

switchFlyState():

- transition: $\text{flying} := \neg\text{flying}$
- exception: none

12 Process Queue Module(M12)

12.1 Module

processQueue

12.2 Uses

[collections.deque](#)

[time](#)

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Types

ProcessQueue = ?

Note: All question marks used here and below mean the type is defined by this module and not eligible to be represented by Python's built-in types.

12.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
ProcessQueue			
enqueue	function, *args		
dequeue			IndexError
process_queue	float		
process_entire_queue			

Note: *args is a sequence of variables of arbitrary types. e.g. $*(x, y) = x, y$

12.4 Semantics

12.4.1 State Variables

queue: sequence of T

12.4.2 State Invariant

None

12.4.3 Assumptions

ProcessQueue() is called before any other access routine

12.4.4 Access Routine Semantics

ProcessQueue():

- transition: $queue := deque()$
- exception: none

enqueue(func, *args):

- transition: $queue := queue.append((func, args))$
- exception: none

dequeue():

- transition: left value removed from queue
 $func, args := queue.popleft()$
 $func(*args)$
- exception: $exc := \neg self.queue \Rightarrow IndexError$

process_queue(maxPeriod):

- transition: $start := time.clock()$
dequeue, while queue $\neg empty \wedge time.clock() - start < maxPeriod$
- exception: none

process_entire_queue():

- transition: dequeue, while queue $\neg empty$
- exception: none

13 Screen Module(M13)

13.1 Module

Screen

13.2 Uses

[math](#)

[pyglet.graphics.TextureGroup](#)

[pyglet.gl](#)

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Types

Screen = ?

13.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
Screen	game, bool		
draw			
mouseClick	int, int, int		
update	float		
mouseMove	int, int, tuple of int, tuple of int		
keyPressed	symbol, int		
keyRelease	symbol, int		
screenResize	int, int		

13.4 Semantics

13.4.1 Environment Variables

Mouse

Keyboard

Monitor

13.4.2 State Variables

game: game

mouseExclusive: bool

13.4.3 State Invariant

None

13.4.4 Assumptions

Screen(game, exclusive) is called before any other access routine

13.4.5 Access Routine Semantics

Screen(game, exclusive):

- transition: game := game
mouseExclusive := exclusive
- exception: None

mouseClick(x, y, button, modifiers):

- transition: pass
Note: Pass used here and below means the extended modules need to have this routine but with different implementations
- exception: None

update(dt):

- transition: pass
- exception: None

mouseMove(x, y, dx, dy):

- transition: pass
- exception: None

keyPressed(symbol, modifiers):

- transition: pass
- exception: None

keyRelease(symbol, modifiers):

- transition: pass

- exception: None

screenResize(width, height):

- transition: pass
- exception: None

draw():

- transition: pass
- exception: None

13.5 Local Functions

_setBGColor(R, G, B, A):

- transition: set background color as *glClearColor(R, G, B, A)*
- exception: $ext := (R < 0 \vee R > 1 \vee G < 0 \vee G > 1 \vee B < 0 \vee B > 1 \vee A < 0 \vee A > 1) \Rightarrow \text{ValueError}$

_setup_fog(R, G, B, A, start, end):

- transition: set up fog with color defined by *glFogfv(R, G, B, A)* and range defined by *glFogf(GL_FOG_START, start)*, *glFogf(GL_FOG_END, end)*
- exception: $ext := (R < 0 \vee R > 1 \vee G < 0 \vee G > 1 \vee B < 0 \vee B > 1 \vee A < 0 \vee A > 1) \Rightarrow \text{ValueError}$

_setup_glbasic():

- transition: basic OpenGL configuration
- exception: None

_setup_2d():

- transition: set up a 2D screen
- exception: None

`_setup_3d()`:

- transition: set up an 3D screen with rotation on z-axis by `glRotatef(x, 0, 1, 0)`, `glRotatef(-y, cos(x), 0, sin(x))`
- exception: None

14 Shape Module(M14)

14.1 Module

Shape3D
Shape2D

14.2 Uses

None

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Types

None

14.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
cube.vertices	float, float, float, float	list of float	
Routine name	In	Out	Exceptions
quad.vertices	float, float, float, float	list of float	

14.4 Semantics

14.4.1 State Variables

None

14.4.2 State Invariant

None

14.4.3 Assumptions

None

14.4.4 Access Routine Semantics

`cube.vertices(x, y, z, n)`:

- output: $out := [x - n, y + n, z - n, x - n, y + n, z + n, x + n, y + n, z + n, x + n, y + n, z - n, x - n, y - n, z - n, x + n, y - n, z - n, x + n, y - n, z + n, x - n, y - n, z + n, x - n, y - n, z - n, x - n, y + n, z + n, x - n, y + n, z - n, x + n, y - n, z + n, x + n, y - n, z - n, x + n, y + n, z - n, x + n, y + n, z + n, x - n, y - n, z + n, x + n, y - n, z + n, x + n, y + n, z + n, x - n, y + n, z + n, x + n, y - n, z - n, x - n, y - n, z - n, x - n, y + n, z - n, x + n, y + n, z - n,]$
- exception: none

`quad.vertices()`:

- output: $out := [x, y, x + w, y, x + w, y + h, x, y + h,]$
- exception: none

15 World Module(M15)

15.1 Module

World

15.2 Uses

sys
time
random
pyglet.image pyglet.gl
pyglet.graphics.TextureGroup
pyglet.window
pyglet.window.key
processQueue.ProcessQueue
shape.Shape3D
loadSource

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Types

World = ?

15.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
World	list of Block, int, int		
skyColor		tuple of float	
changeMode	str		ValueError
updateWorld	float, Player		
clearWorld			
changeWorld	World		ValueError
setupWorld			
collide	tuple of float, Creature	tuple of float	
hit_test	tuple of float ×2, int	tuple of float ×2	
add_block	tuple of float, Block, bool		ValueError
remove_block	tuple of float, bool		ValueError
show_block	tuple of float, bool		IndexError, ValueError
hide_block	tuple of float , bool		IndexError

15.4 Semantics

15.4.1 State Variables

batch: Batch
world: World
shown: {tuple of float: Block}
_shown: {tuple of float: Batch}
sectors: {tuple of int: [tuple of int]}
sectorSize: int
processQueue: ProcessQueue
gravity: int
sector: tuple of int
mode: str
blocks: str: block

15.4.2 State Invariant

None

15.4.3 Assumptions

World() is called before any other access routine, and always call setupWprld() after calling World().

15.4.4 Access Routine Semantics

World(allBlocks, sectorSize = 16, gravity = 20):

- transition: batch := pygame.graphics.Batch()
world, shown, _shown, sectors := {}, {}, {}, {}
sectorSize := sectorSize
processQueue := ProcessQueue()
gravity := gravity
sector := Null
mode := "day"
blocks := name: block in allBlocks
- exception: None

skyColor():

- output:

	<i>out</i> :=
self.mode = "day"	(0.5, 0.69, 1.0, 1)
self.mode = "night"	(0.05, 0, 0.15, 1)

- exception: None

changeMode(mode):

- transition: $\text{self.mode} := \text{mode}$, if $(\text{mode} = \text{"day"} \vee \text{mode} = \text{"night"})$
- exception: $\text{exc} := (\text{mode} \neq \text{"day"} \wedge \text{mode} \neq \text{"night"}) \Rightarrow \text{ValueError}$

updateWorld(freshPeriod, player):

- transition:

processQueue.process_queue(1.0/freshPeriod)	
sector := _sectorize(player.position)	
sector \neq self.sector	_change_sectors(self.sector, sector)
self.sector = Null	processQueue.process_entire_queue()
self.sector := sector	

- exception: None

setupWorld():

- transition: position := (0,0,0)
world.clearWorld()
world.setupWorld()
changeScene("game")
- exception: None

changeScene():

- transition:

n, s, y := 80, 1, 0		
for x in $(-n, n + 1)$:	add_block((x, y - 2, z), GRASS, False)	
for z in $(-n, n + 1)$:	add_block((x, y - 2, z), GRASS, False)	
$(x \vee z) \in (-n, n)$	for dy in $(-2, 3)$:	add_block((x, y + dy, z), MARBLE, False)
o := n - 10		
for _ in 120:	a, b, c, d := random(-o, o), random(-o, o), -1, 1 h, s := random(1, 6), random(4, 8) t := random([GRASS, STONE, BRICK])	
	for y in $(c, c + h)$:	$(x - a)^2 + (z - b)^2 > (s + 1)^2$ continue
	for x in $(a - s, a + s + 1)$:	$(x - 0)^2 + (z - 0)^2 < 5^2$ continue
	for z in $(b - s, b + s + 1)$:	add_block((x, y, z), t.name, False)
	s := s - d	

- exception: None

collide(position, creature):

- transition:

pad, p, np := 0.25, list(position), _normalize(position)		
for face in allFaces:	\neg face[i]:	continue
for i in (3):	$d := (p[i] - np[i]) \times face[i]$	
	$d < pad$	continue
for dy in (creature.height):	op := list(np) $op[1] := op[1] - dy$ $op[i] := op[i] + face[i]$	
	tuple(op) \notin self.world:	continue
	$p[i] := p[i] - (d - pad) \times face[i]$	
	face = (0, -1, 0) \vee (0, 1, 0)	creature.dy := 0
	break	

- output: $out := tuple(p)$
- exception: None

hit_test(position, vector, max_distance=8):

- transition and output:

		$out :=$
m := 8; x, y, z := position; dx, dy, dz := vector; previous := None		
for _ in (max_distance \times m):	key := _normalize((x, y, z))	
	key \neq previous \wedge key \in self.world	key, previous
	previous := key $x, y, z := x + dx/m, y + dy/m, z + dz/m$	
		Null, Null

- exception: None

add_block(position, block, immediate=True):

- transition:

position \in self.world	remove_block(position, immediate)
world[position] := block	
sectors.setdefault(_sectorize(position), []).append(position)	
immediate	_exposed(position) show_block(position)
	_check_neighbors(position)

- exception:

	$exc :=$
block \notin self.blocks	ValueError

remove_block(position, immediate=True):

- transition:

del self.world[position]		
sectors[_sectorize(position)].remove(position)		
immediate	position ∈ shown	hide_block(position)
	_check_neighbors(position)	

- exception:

	<i>exc</i> :=
block ∉ self.world	ValueError

show_block(position, immediate=True):

- transition:

immediate	block := self.blocks[self.world[position]] shown[position] := block x, y, z := position vertex_data := Shape3D.cube_vertices(x, y, z, 0.5) texture_data := list(block.coordinates) _shown[position] := batch.add(block.texture, vertex_data, texture_data)
¬ immediate	processQueue.enqueue(show_block, position, True)

- exception:

		<i>exc</i> :=
immediate	position ∉ self.blocks world[position] ∉ self.blocks	IndexError ValueError

hide_block(position, immediate=True):

- transition:

immediate	shown.pop(position) _shown.pop(position).delete()
¬ immediate	processQueue.enqueue(hide_block, position, True)

- exception:

		<i>exc</i> :=
immediate	position ∉ shown	IndexError

15.5 Local Functions

_check_neighbors(position):

- transition:

x, y, z := position			
for dx, dy, dz key := (x + dx, y + dy, z + dz)			
in allFaces:	key ∉ self.world	continue	
	._exposed(key)	key ∉ shown	show_block(key)
	¬ ._exposed(key)	key ∈ shown	hide_block(key)

- exception: None

._exposed(position):

- output:

			<i>out :=</i>
x, y, z := position			
for dx, dy, dz	(x + dx, y + dy, z + dz) ∉ self.world	True	
in allFaces:			
			False

- exception: None

._show_sector(sector):

- transition:

for position in sectors.get(sector, [])	position ∉ shown ∧ self._exposed(position)	show_block(position, False):
--	---	------------------------------

- exception: None

._hide_sector(sector):

- transition:

for position in sectors.get(sector, [])	position ∈ shown	hide_block(position, False)
--	------------------	-----------------------------

- exception: None

._change_sectors(before, after):

- transition:

before_set, after_set, pad := set(), set(), 4		
for dx in (-pad, pad + 1):	$dx^2 + dy^2 + dz^2 > (pad + 1)^2$	continue
for dy in [0]:		
for dz in (-pad, pad + 1):	before	x, y, z := before before_set.add((x + dx, y + dy, z + dz))
	after	x, y, z := after after_set.add((x + dx, y + dy, z + dz))
show, hide := before_set, after_set		
for sector in show:	_show_sector(sector)	
for sector in hide:	_hide_sector(sector)	

- exception: None

_sectorize(position):

- output: x, y, z := _normalize(position)
x, y, z := x//sectorSize, y//sectorSize, z//sectorSize
out := (x, 0, z)
- exception: None

_sectorize(position):

- output: x, y, z := position
x, y, z := (int(round(x)), int(round(y)), int(round(z)))
out := (x, y, z)
- exception: None