serge Documentation

Release 0.4

Paul Paterson

July 14, 2012

CONTENTS

Documentation Overview:

CHAPTER

ONE

TUTORIALS

1.1 Tutorial 1: Snake Part 1

Contents

1.1.1 Introduction

In this tutorial we will explore some of the basic features of serge. We will cover *Worlds*, *Zones*, and *Actors* using a simple snake game as an example.

The code for the game is presented as a single file and at each stage the entire code is show. You can simply copy the code sample into one file and then execute it to run the game. As we add graphics and sounds you will need to download the game assets. You can download a ZIP file here. This contains the folders and files needed.

1.1.2 Game

The game we will be building is a simple Snake game. In this game you control an ever growing snake. You turn the snake head left and right and try to avoid the head hitting the rest of the body.

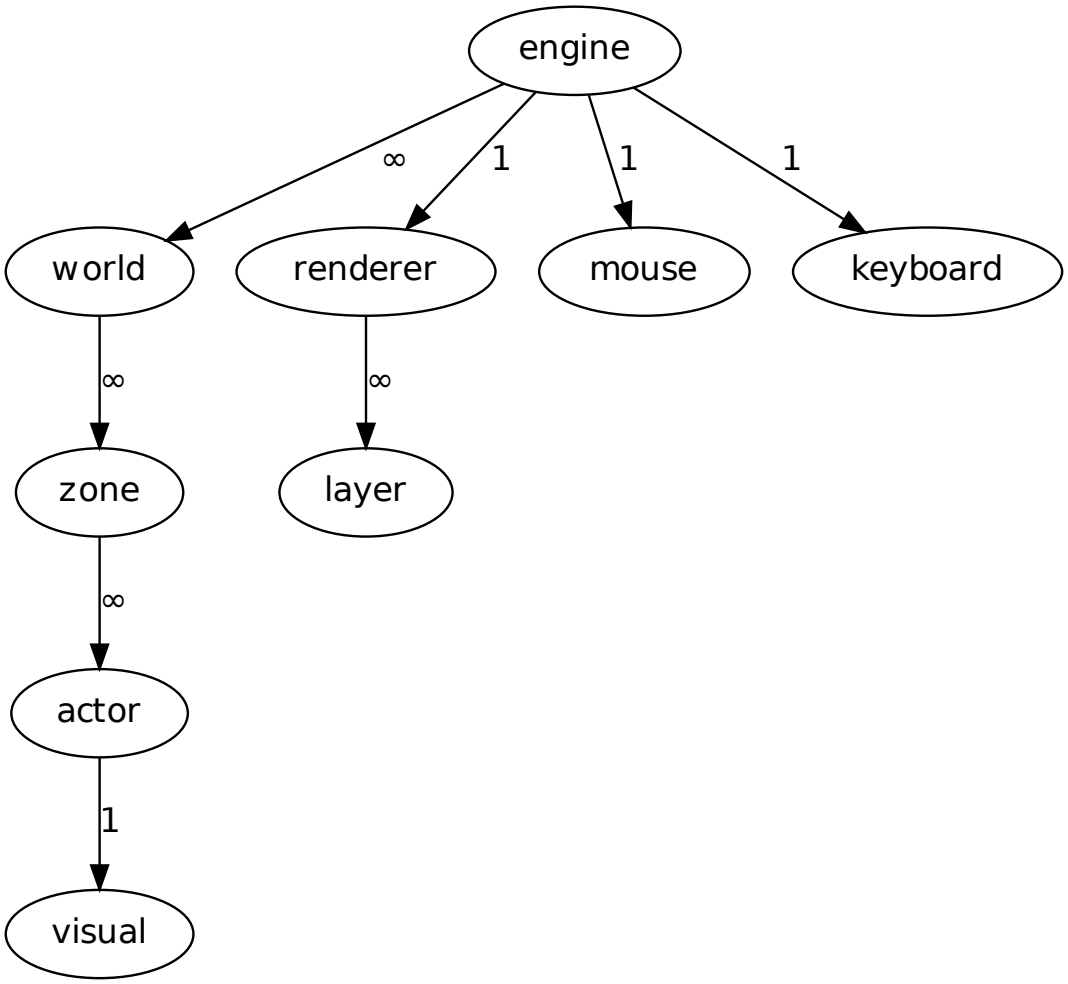
We are going to start building the game with simple graphics and then, when the core game is working we will build in sprites, sounds, and fonts to make it look more attractive.

1.1.3 Engine Overview

The basic structure of a serge game revolves around the main *Engine*. The engine runs the game.

Most of what happens in a game, the logic etc, is controlled by various actors. Actors live in worlds and worlds are divided into zones. Actors have visuals which are how they appear to the player.

Finally, the engine has a renderer, which handles drawing the actor's visuals to the screen. Rendering happens on one or more layers. This allows you to control which actors are painted in front of which, eg to put the player in front of the background.



1.1.4 Setting Things Up Quickly

You have a fair amount of flexibility in how you configure your engine, but for this example we will use a simple default setup which includes,

- a single world, called *lab*, with a single zone

- three rendering layers (called *back*, *middle* and *front*)

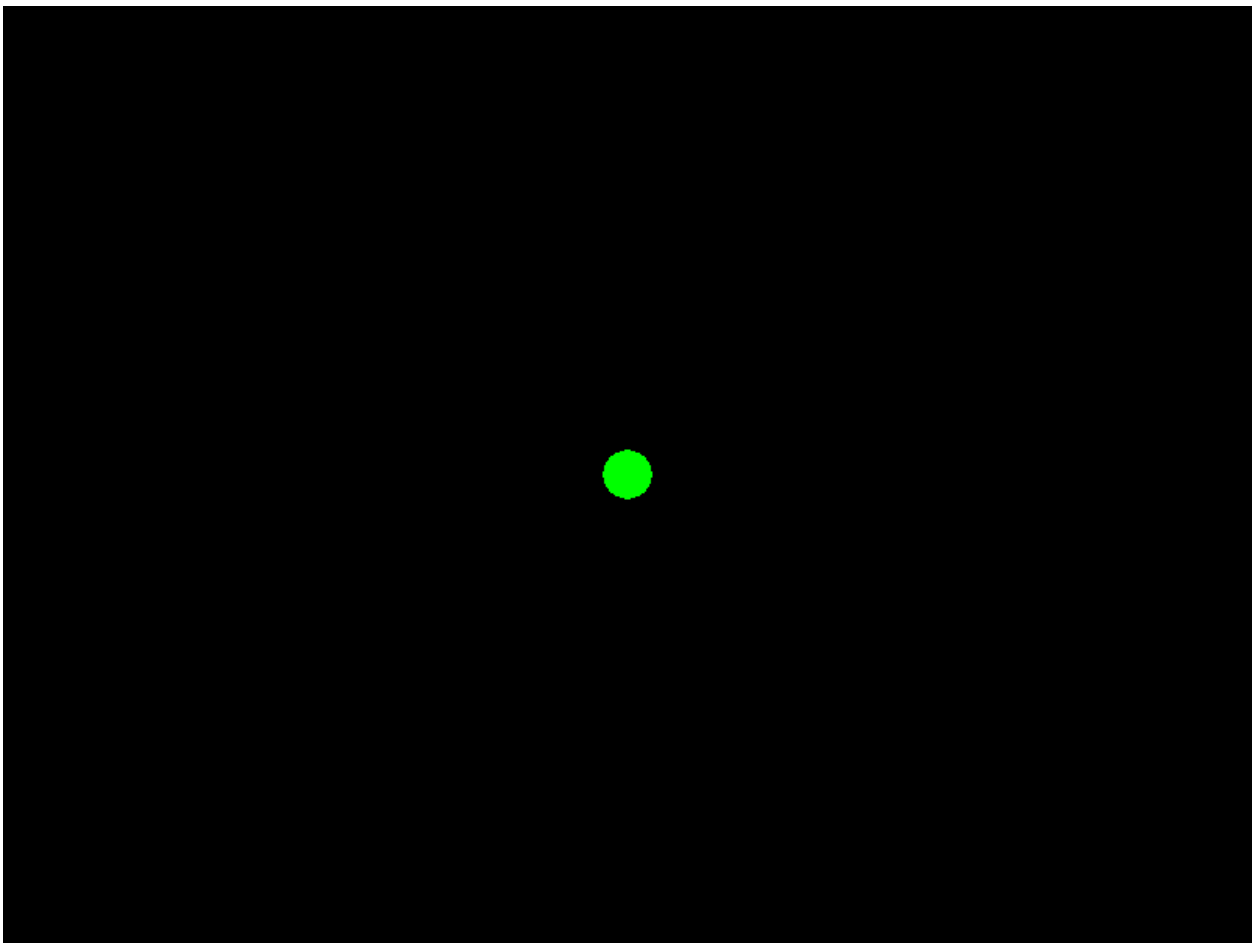
Here is the code:

```
import serge.blocks.utils

engine = serge.blocks.utils.getSimpleSetup(800, 600)
engine.run(60)
```

When you run this you should see some logging appear in the console and then a blank window will appear.

The *getSimpleSetup* call returns an engine which will render to a screen of size 800x600. The call to *engine.run* then runs the engine at 60 frames per second.

1.1.5 Adding The Snake's Head



Ok, let's now add the head of the snake. There will be a fail amount of logic associated with the snake so we normally want to encapsulate this in an actor. To begin with we will create the head as a simple circle using the following code.

```
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils

class Snake(serge.actor.Actor):
    """Represents the snake"""

```

```
    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
        self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
        self.setLayerName('middle')

# Create the engine
engine = serge.blocks.utils.getSimpleSetup(800, 600)
world = engine.getWorld('lab')

# Create the snake
snake = Snake()
world.addActor(snake)
snake.moveTo(400, 300)

# Run the game
engine.run(60)
```

When you run this you should see a green circle in the middle of the screen. This is the snakes head. The graphic comes from the line:

```
self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
```

This sets the visual representation of the actor to be a green circle with a radius of 16 pixels. Later on we will replace this with a sprite but for now let's keep it simple.

1.1.6 Moving The Head

Now it is time to move the snake's head.

The engine will call the *updateActor* method on an actors in the currently active world every timestep. This is the normal way that we perform any game logic and so we will use it to move the snake.

We need to give the snake a certain direction, which we set up in the *__init__* method. There is a *block* that we can make use of for cardinal directions.

```
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils
import serge.blocks.directions

class Snake(serge.actor.Actor):
    """Represents the snake"""

    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
        self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
        self.setLayerName('middle')
        self.current_direction = serge.blocks.directions.N

    def updateActor(self, interval, world):
        """Update the snake"""
        super(Snake, self).updateActor(interval, world)
        #
        offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
        self.move(*offset)

```

```
# Create the engine
engine = serge.blocks.utils.getSimpleSetup(800, 600)
world = engine.getWorld('lab')

# Create the snake
snake = Snake()
world.addActor(snake)
snake.moveTo(400, 300)

# Run the game
engine.run(60)
```

Now the snake's head should gradually move up the screen. This is going *north* because we chose this as the current direction in the *__init__* method. The *getVectorFromCardinal* function returns a *Vec2d* so we can multiply it by any constant to create the right length. You can experiment with the number 5 to adjust the difficulty of the game.

Note Always remember to call the base class methods (ie using *super*) for methods like *updateActor*.

1.1.7 Interacting With The Snake

So far the snake is just going forward with no input from the user. Let's now allow the user to move the head around. We do this by looking for keyboard input.

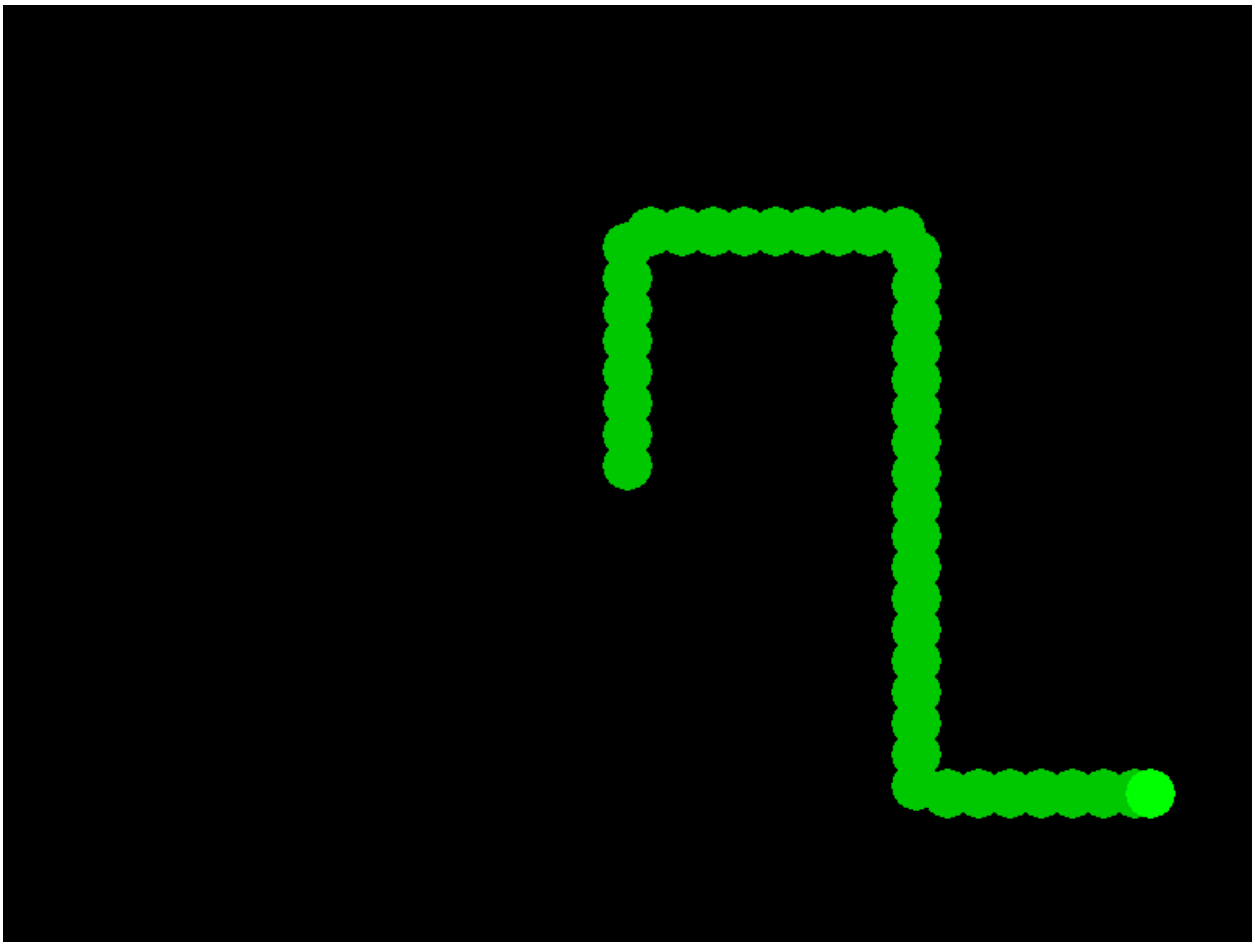
The engine has a keyboard object and we can use this to check. Note that for efficiency it is best to get hold of the keyboard object and anything else you may need in the *addedToWorld* method of an actor. This method is called just after the actor is added to the world and is a great place to do initialisation. It is usually better to do things here rather than in the *__init__* method because at *__init__* you do not know anything about the world you are in.

```
import pygame

import serge.engine
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils
import serge.blocks.directions

class Snake(serge.actor.Actor):
    """Represents the snake"""

    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
        self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
        self.setLayerName('middle')
        self.current_direction = serge.blocks.directions.N

    def addedToWorld(self, world):
        """The snake was added to the world"""
        super(Snake, self).addedToWorld(world)
        #
        self.keyboard = serge.engine.CurrentEngine().getKeyboard()

    def updateActor(self, interval, world):
        """Update the snake"""
        super(Snake, self).updateActor(interval, world)
        #
        # Move the head
```

```
30          if self.keyboard.isClicked(pygame.K_LEFT):
31              rotation = +90
32          elif self.keyboard.isClicked(pygame.K_RIGHT):
33              rotation = -90
34          else:
35              rotation = 0
36          #
37          # Change direction
38          if rotation:
39              current_angle = serge.blocks.directions.getAngleFromCardinal(self.current_direction)
40              self.current_direction = serge.blocks.directions.getCardinalFromAngle(current_angle+rotat
41          #
42          # Move
43          offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
44          self.move(*offset)
45
46  # Create the engine
47  engine = serge.blocks.utils.getSimpleSetup(800, 600)
48  world = engine.getWorld('lab')
49
50  # Create the snake
51  snake = Snake()
52  world.addActor(snake)
53  snake.moveTo(400, 300)
54
55  # Run the game
56  engine.run(60)
```

You should now be able to direct the snake's head using the left and right arrow keys. Notice that we use the *isClicked* method of the keyboard. This means that the user has to press and release the key before we will turn the snake. We will see later that we can use the *isDown* to create a different feel to the game.

1.1.8 Leaving A Trail



So far the snake is just a head. Let's add a body to it now.

The body of the snake will be made up of a series of segments. We should lay a new segment down each time we have moved a certain distance. However, we cannot just count up how far the head has gone since the player may change direction at any time.

So the algorithm is:

- Add a new segment to begin with
- Each iteration check if adding a new segment would overlap the last
- If it overlaps do nothing
- It if doesn't overlap then add it

Let's look at the code.

```
import pygame

import serge.engine
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils
import serge.blocks.directions

```

```
class Snake(serge.actor.CompositeActor):
    """Represents the snake"""

    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
        self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
        self.setLayerName('middle')
        self.current_direction = serge.blocks.directions.N

    def addedToWorld(self, world):
        """The snake was added to the world"""
        super(Snake, self).addedToWorld(world)
        #
        self.keyboard = serge.engine.CurrentEngine().getKeyboard()

    def updateActor(self, interval, world):
        """Update the snake"""
        super(Snake, self).updateActor(interval, world)
        #
        # Move the head
        if self.keyboard.isClicked(pygame.K_LEFT):
            rotation = +90
        elif self.keyboard.isClicked(pygame.K_RIGHT):
            rotation = -90
        else:
            rotation = 0
        #
        # Change direction
        if rotation:
            current_angle = serge.blocks.directions.getAngleFromCardinal(self.current_direction)
            self.current_direction = serge.blocks.directions.getCardinalFromAngle(current_angle+rotat
        #
        # Move
        offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
        self.move(*offset)
        #
        # Add a new segment if needed
        if not self.getChildren() or self.getDistanceFrom(self.getChildren()[-1]) > 16:
            self.addSegment()

    def addSegment(self):
        """Add a new body segment"""
        segment = serge.actor.Actor('segment')
        segment.visual = serge.blocks.visualblocks.Circle(16, (0,200,0))
        segment.setLayerName('middle')
        segment.moveTo(self.x, self.y)
        self.addChild(segment)


# Create the engine
engine = serge.blocks.utils.getSimpleSetup(800, 600)
world = engine.getWorld('lab')

# Create the snake
snake = Snake()
world.addActor(snake)
snake.moveTo(400, 300)
```

```
67
68 # Run the game
69 engine.run(60)
```

Notice on line 9 we changed the *Actor* to a *CompositeActor*. A *CompositeActor* is just like an actor but it can have child actors also. This helps keep track of the segments and means that when we add a segment as a child it will be added to the same world.

We check the distance from the last child using the *getDistanceFrom* method. You can try different values than 16 to play around with how the tail looks.

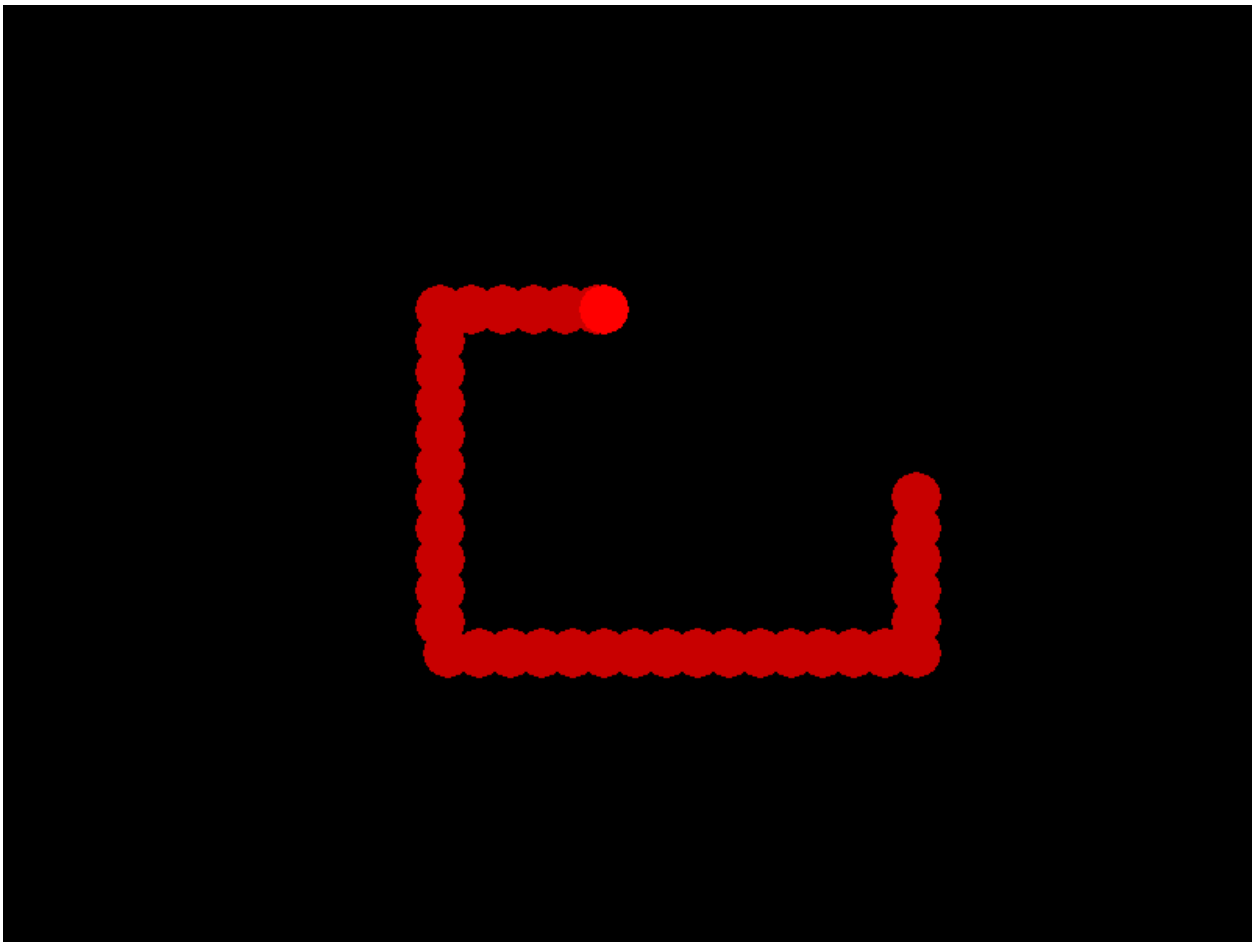
1.1.9 Hitting The Body

So far the game is easy. You can run over the tail as many times as you like. So now let's make that kill the snake.

We can use the *getDistanceFrom* method of the head to check if it ever collides with a part of the body.

```
1  import pygame
2
3  import serge.engine
4  import serge.actor
5  import serge.blocks.visualblocks
6  import serge.blocks.utils
7  import serge.blocks.directions
8
9  class Snake(serge.actor.CompositeActor):
10     """Represents the snake"""
11
12     def __init__(self):
13         """Initialise the snake"""
14         super(Snake, self).__init__('snake', 'snake-head')
15         self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
16         self.setLayerName('middle')
17         self.current_direction = serge.blocks.directions.N
18
19     def addedToWorld(self, world):
20         """The snake was added to the world"""
21         super(Snake, self).addedToWorld(world)
22         #
23         self.keyboard = serge.engine.CurrentEngine().getKeyboard()
24
25     def updateActor(self, interval, world):
26         """Update the snake"""
27         super(Snake, self).updateActor(interval, world)
28         #
29         # Move the head
30         if self.keyboard.isClicked(pygame.K_LEFT):
31             rotation = +90
32         elif self.keyboard.isClicked(pygame.K_RIGHT):
33             rotation = -90
34         else:
35             rotation = 0
36         #
37         # Change direction
38         if rotation:
39             current_angle = serge.blocks.directions.getAngleFromCardinal(self.current_direction)
40             self.current_direction = serge.blocks.directions.getCardinalFromAngle(current_angle+rotat
41         #
```

```
42          # Move
43          offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
44          self.move(*offset)
45          #
46          # Add a new segment if needed
47          if not self.getChildren() or self.getDistanceFrom(self.getChildren()[-1]) > 16:
48              self.addSegment()
49          #
50          # Check if we hit the body
51          if self.hitBody():
52              self.initiateDeathAnimation()
53
54      def addSegment(self):
55          """Add a new body segment"""
56          segment = serge.actor.Actor('segment')
57          segment.visual = serge.blocks.visualblocks.Circle(16, (0,200,0))
58          segment.setLayerName('middle')
59          segment.moveTo(self.x, self.y)
60          self.addChild(segment)
61
62      def hitBody(self):
63          """Return True if the head has hit the body
64
65          Look to see if we overlap with any body segment except the last
66          (we are allowed to overlap the last since we just put it down)
67
68          """
69          for segment in self.getChildren()[:-1]:
70              if self.getDistanceFrom(segment) < 16:
71                  return True
72          return False
73
74      def initiateDeathAnimation(self):
75          """Begin showing the death of the snake"""
76          self.log.info('Snake died!')
77
78
79  # Create the engine
80  engine = serge.blocks.utils.getSimpleSetup(800, 600)
81  world = engine.getWorld('lab')
82
83  # Create the snake
84  snake = Snake()
85  world.addActor(snake)
86  snake.moveTo(400, 300)
87
88  # Run the game
89  engine.run(60)
```



Right now the snake actually doesn't die it just logs to the console. For the death animation, lets turn the body red and then gradually remove body parts until we get to the head.

Now for this kind of animation you want it to take a while to show. If you implement this in the *upateActor* method then it might happen pretty quickly since that is called 60 times a second. We can use a timed callback to do this where we can control how often it is called.

Callbacks fall under a category called *behaviours*. These are generally useful kinds of activities that can apply to any actors. To utilize *behaviours* you create a *BehaviourManager*, which is just a special actor, and then use it to assign behaviours.

```
import pygame

import serge.engine
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils
import serge.blocks.directions
import serge.blocks.behaviours

class Snake(serge.actor.CompositeActor):
    """Represents the snake"""

    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
```

```
16          self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
17          self.setLayerName('middle')
18          self.current_direction = serge.blocks.directions.N
19          self.is_dying = False
20
21      def addedToWorld(self, world):
22          """The snake was added to the world"""
23          super(Snake, self).addedToWorld(world)
24          #
25          self.keyboard = serge.engine.CurrentEngine().getKeyboard()
26          self.manager = serge.blocks.behaviours.BehaviourManager('manager', 'behaviour-manager')
27          world.addActor(self.manager)
28
29      def updateActor(self, interval, world):
30          """Update the snake"""
31          super(Snake, self).updateActor(interval, world)
32          #
33          # Move the head
34          if self.keyboard.isClicked(pygame.K_LEFT):
35              rotation = +90
36          elif self.keyboard.isClicked(pygame.K_RIGHT):
37              rotation = -90
38          else:
39              rotation = 0
40          #
41          # Change direction
42          if rotation:
43              current_angle = serge.blocks.directions.getAngleFromCardinal(self.current_direction)
44              self.current_direction = serge.blocks.directions.getCardinalFromAngle(current_angle+rotat
45          #
46          # Move
47          if not self.is_dying:
48              offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
49              self.move(*offset)
50              #
51              # Add a new segment if needed
52              if not self.getChildren() or self.getDistanceFrom(self.getChildren()[-1]) > 16:
53                  self.addSegment()
54              #
55              # Check if we hit the body
56              if self.hitBody():
57                  self.initiateDeathAnimation()
58
59      def addSegment(self):
60          """Add a new body segment"""
61          segment = serge.actor.Actor('segment')
62          segment.visual = serge.blocks.visualblocks.Circle(16, (0,200,0))
63          segment.setLayerName('middle')
64          segment.moveTo(self.x, self.y)
65          self.addChild(segment)
66
67      def hitBody(self):
68          """Return True if the head has hit the body
69
70          Look to see if we overlap with any body segment except the last
71          (we are allowed to overlap the last since we just put it down)
72
73          """
```

```
74          for segment in self.getChildren()[:-1]:
75              if self.getDistanceFrom(segment) < 16:
76                  return True
77          return False
78
79      def initiateDeathAnimation(self):
80          """Begin showing the death of the snake"""
81          self.log.info('Snake died!')
82          self.animation = self.manager.assignBehaviour(self,
83              serge.blocks.behaviours.TimedCallback(1000/len(self.getChildren()), self.removeTail), 'de
84          self.is_dying = True
85          for segment in self.getChildren():
86              segment.visual.colour = (200, 0, 0)
87          self.visual.colour = (255, 0, 0)
88
89      def removeTail(self, world, actor, interval):
90          """Remove part of the tail"""
91          self.log.debug('Removing part of the tail')
92          if self.getChildren():
93              self.removeChild(self.getChildren()[0])
94          else:
95              self.animation.markComplete()
96
97  # Create the engine
98  engine = serge.blocks.utils.getSimpleSetup(800, 600)
99  world = engine.getWorld('lab')
100
101 # Create the snake
102 snake = Snake()
103 world.addActor(snake)
104 snake.moveTo(400, 300)
105
106 # Run the game
107 engine.run(60)
```

There is a bit of housekeeping we have to do here. On line 19 we create a new property *is_dying* which we will set when the snake is dying. When this is true we do not want to move the snake, add bodies or check for death (again) so we protect some of the lines in the *updateActor* method to prevent them being called.

The reset of the updates create the *manager* and then use it to assign the *callback*. We call the *removeTail* method to gradually remove our children. The time interval is set to make sure if takes about a second to remove the whole tail. When it is done it calls the *markComplete* method on the *behaviour* to tell the engine that it can be discarded as we wont need it again.

1.1.10 Restarting The Game

There isn't anything to do after you die now so let's add some text and a way to restart.

```
import pygame

import serge.engine
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils
import serge.blocks.directions
import serge.blocks.behaviours

class Snake(serge.actor.CompositeActor):
    """Represents the snake"""

    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
        self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
        self.setLayerName('middle')
        self.current_direction = serge.blocks.directions.N
        self.is_dying = False

    def addedToWorld(self, world):
        """The snake was added to the world"""
```

```
23          super(Snake, self).addedToWorld(world)
24          #
25          self.keyboard = serge.engine.CurrentEngine().getKeyboard()
26          self.manager = serge.blocks.behaviours.BehaviourManager('manager', 'behaviour-manager')
27          world.addActor(self.manager)
28          #
29          self.restart_text = serge.blocks.utils.addVisualActorToWorld(world, 'text', 'restart',
30              serge.visual.Text('Game Over - Press ENTER to restart', (255, 255, 255), font_size=20),
31              layer_name='front',
32              center_position=(400, 300))
33          self.restart_text.visible = False
34
35      def updateActor(self, interval, world):
36          """Update the snake"""
37          super(Snake, self).updateActor(interval, world)
38          #
39          # Move the head
40          if self.keyboard.isClicked(pygame.K_LEFT):
41              rotation = +90
42          elif self.keyboard.isClicked(pygame.K_RIGHT):
43              rotation = -90
44          else:
45              rotation = 0
46          #
47          # Change direction
48          if rotation:
49              current_angle = serge.blocks.directions.getAngleFromCardinal(self.current_direction)
50              self.current_direction = serge.blocks.directions.getCardinalFromAngle(current_angle+rotat
51          #
52          # Move
53          if not self.is_dying:
54              offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
55              self.move(*offset)
56              #
57              # Add a new segment if needed
58              if not self.getChildren() or self.getDistanceFrom(self.getChildren()[-1]) > 16:
59                  self.addSegment()
60              #
61              # Check if we hit the body
62              if self.hitBody():
63                  self.initiateDeathAnimation()
64          elif self.animation.isComplete():
65              if self.keyboard.isClicked(pygame.K_KP_ENTER) or self.keyboard.isClicked(pygame.K_RETURN)
66                  self.restartGame()
67
68      def addSegment(self):
69          """Add a new body segment"""
70          segment = serge.actor.Actor('segment')
71          segment.visual = serge.blocks.visualblocks.Circle(16, (0,200,0))
72          segment.setLayerName('middle')
73          segment.moveTo(self.x, self.y)
74          self.addChild(segment)
75
76      def hitBody(self):
77          """Return True if the head has hit the body
78
79          Look to see if we overlap with any body segment except the last
80          (we are allowed to overlap the last since we just put it down)
```

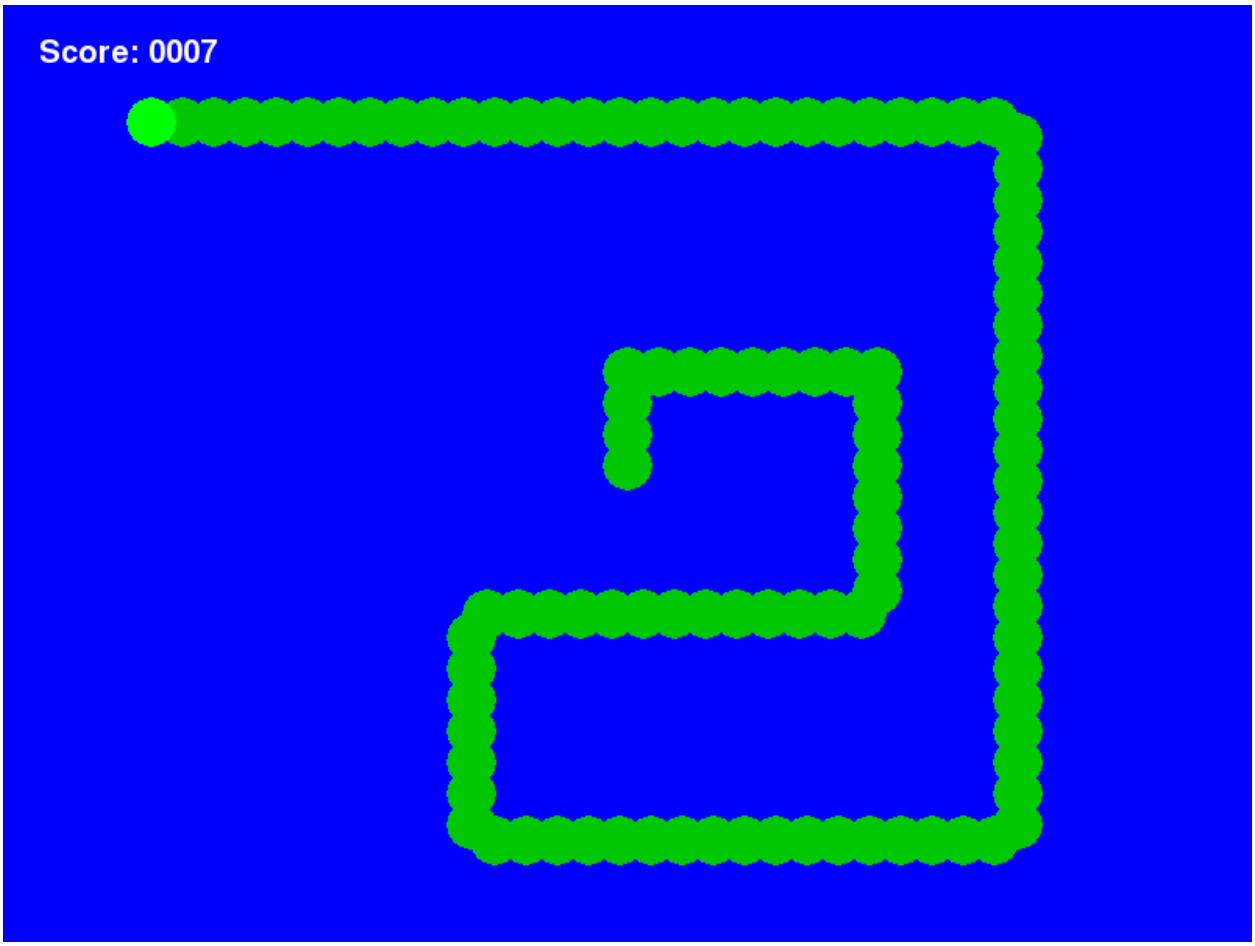
```
81
82          """
83          for segment in self.getChildren()[:-1]:
84              if self.getDistanceFrom(segment) < 16:
85                  return True
86          return False
87
88      def initiateDeathAnimation(self):
89          """Begin showing the death of the snake"""
90          self.log.info('Snake died!')
91          self.animation = self.manager.assignBehaviour(self,
92              serge.blocks.behaviours.TimedCallback(1000/len(self.getChildren()), self.removeTail), 'de
93          self.is_dying = True
94          for segment in self.getChildren():
95              segment.visual.colour = (200, 0, 0)
96          self.visual.colour = (255, 0, 0)
97
98      def removeTail(self, world, actor, interval):
99          """Remove part of the tail"""
100         self.log.debug('Removing part of the tail')
101         if self.getChildren():
102             self.removeChild(self.getChildren()[0])
103         else:
104             self.animation.markComplete()
105             self.restart_text.visible = True
106
107     def restartGame(self):
108         """Restart the game"""
109         self.is_dying = False
110         self.restart_text.visible = False
111         self.visual.colour = (0,255,0)
112         self.current_direction = serge.blocks.directions.N
113         self.moveTo(400, 300)
114
115 # Create the engine
116 engine = serge.blocks.utils.getSimpleSetup(800, 600)
117 world = engine.getWorld('lab')
118
119 # Create the snake
120 snake = Snake()
121 world.addActor(snake)
122 snake.moveTo(400, 300)
123
124 # Run the game
125 engine.run(60)
```

We create some text in the *addedToWorld* method. Note how we use the *front* layer to make sure that the text appears before anything else on the screen. We set the *visible* property to *False* initially because we do not want it to show until the end of the game.

Then in the *updateActor* method we check for the keypress when we are dying *and* when the animation has completed. We do not want to allow the user to press enter before the snake is completely cleaned up.

When the user does press enter then we use the *restartGame* method to clean up all the flags and this starts everything over again.

1.1.11 Minor Polishing



Ok, let's take a bit of time polish things up a bit here with a number of changes.

- Add a background to the display
- Allow the user to press ESCAPE to quit the game at any time
- Kill the snake if it goes off the screen
- Keep score of how long the user survived

```
import pygame

import serge.engine
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils
import serge.blocks.directions
import serge.blocks.behaviours
import serge.blocks.actors

class Snake(serge.actor.CompositeActor):
    """Represents the snake"""

    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
```

```
17          self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
18          self.setLayerName('middle')
19          self.current_direction = serge.blocks.directions.N
20          self.is_dying = False
21
22      def addedToWorld(self, world):
23          """The snake was added to the world"""
24          super(Snake, self).addedToWorld(world)
25          #
26          self.keyboard = serge.engine.CurrentEngine().getKeyboard()
27          self.manager = serge.blocks.behaviours.BehaviourManager('manager', 'behaviour-manager')
28          world.addActor(self.manager)
29          #
30          # Text to display when the game is over
31          self.restart_text = serge.blocks.utils.addVisualActorToWorld(world, 'text', 'restart',
32              serge.visual.Text('Game Over - Press ENTER to restart', (255, 255, 255), font_size=20),
33              layer_name='front',
34              center_position=(400, 300))
35          self.restart_text.visible = False
36          #
37          # A background for the game
38          self.bg = serge.blocks.utils.addVisualActorToWorld(world, 'bg', 'bg',
39              serge.blocks.visualblocks.Rectangle((800, 600), (0,0,255)),
40              layer_name='back',
41              center_position=(400, 300))
42          #
43          # Text to show the score
44          self.score = serge.blocks.utils.addActorToWorld(world,
45              serge.blocks.actors.NumericText('text', 'score', 'Score: %04d',
46                  (255, 255, 255), font_size=20, value=0, align='left'),
47              layer_name='front',
48              center_position=(80, 30))
49
50      def updateActor(self, interval, world):
51          """Update the snake"""
52          super(Snake, self).updateActor(interval, world)
53          #
54          # Quit if requested
55          if self.keyboard.isClicked(pygame.K_ESCAPE):
56              serge.engine.CurrentEngine().stop()
57          #
58          # Move the head
59          if self.keyboard.isClicked(pygame.K_LEFT):
60              rotation = +90
61          elif self.keyboard.isClicked(pygame.K_RIGHT):
62              rotation = -90
63          else:
64              rotation = 0
65          #
66          # Change direction
67          if rotation:
68              current_angle = serge.blocks.directions.getAngleFromCardinal(self.current_direction)
69              self.current_direction = serge.blocks.directions.getCardinalFromAngle(current_angle+rotat
70          #
71          # Move
72          if not self.is_dying:
73              offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
74              self.move(*offset)
```

```
75              #
76              # Add a new segment if needed
77              if not self.getChildren() or self.getDistanceFrom(self.getChildren()[-1]) > 16:
78                  self.addSegment()
79              #
80              # Check if we hit the body
81              if self.hitBody() or self.offScreen():
82                  self.initiateDeathAnimation()
83              #
84              # Increase score
85              self.score.value += interval/1000.0
86          elif self.animation.isComplete():
87              if self.keyboard.isClicked(pygame.K_KP_ENTER) or self.keyboard.isClicked(pygame.K_RETURN)
88                  self.restartGame()
89
90      def addSegment(self):
91          """Add a new body segment"""
92          segment = serge.actor.Actor('segment')
93          segment.visual = serge.blocks.visualblocks.Circle(16, (0,200,0))
94          segment.setLayerName('middle')
95          segment.moveTo(self.x, self.y)
96          self.addChild(segment)
97
98      def hitBody(self):
99          """Return True if the head has hit the body
100
101         Look to see if we overlap with any body segment except the last
102         (we are allowed to overlap the last since we just put it down)
103
104         """
105         for segment in self.getChildren()[:-1]:
106             if self.getDistanceFrom(segment) < 16:
107                 return True
108         return False
109
110     def offScreen(self):
111         """Return True if we are off the screen"""
112         return self.x < 0 or self.x > 800 or self.y < 0 or self.y > 600
113
114     def initiateDeathAnimation(self):
115         """Begin showing the death of the snake"""
116         self.log.info('Snake died!')
117         self.animation = self.manager.assignBehaviour(self,
118             serge.blocks.behaviours.TimedCallback(1000/len(self.getChildren()), self.removeTail), 'de
119         self.is_dying = True
120         for segment in self.getChildren():
121             segment.visual.colour = (200, 0, 0)
122         self.visual.colour = (255, 0, 0)
123
124     def removeTail(self, world, actor, interval):
125         """Remove part of the tail"""
126         self.log.debug('Removing part of the tail')
127         if self.getChildren():
128             self.removeChild(self.getChildren()[0])
129         else:
130             self.animation.markComplete()
131             self.restart_text.visible = True
132
```

```
133     def restartGame(self):
134         """Restart the game"""
135         self.is_dying = False
136         self.restart_text.visible = False
137         self.visual.colour = (0,255,0)
138         self.current_direction = serge.blocks.directions.N
139         self.score.value = 0
140         self.moveTo(400, 300)
141
142 # Create the engine
143 engine = serge.blocks.utils.getSimpleSetup(800, 600)
144 world = engine.getWorld('lab')
145
146 # Create the snake
147 snake = Snake()
148 world.addActor(snake)
149 snake.moveTo(400, 300)
150
151 # Run the game
152 engine.run(60)
```

This tutorial continues in *Tutorial 2: Snake Part 2*.

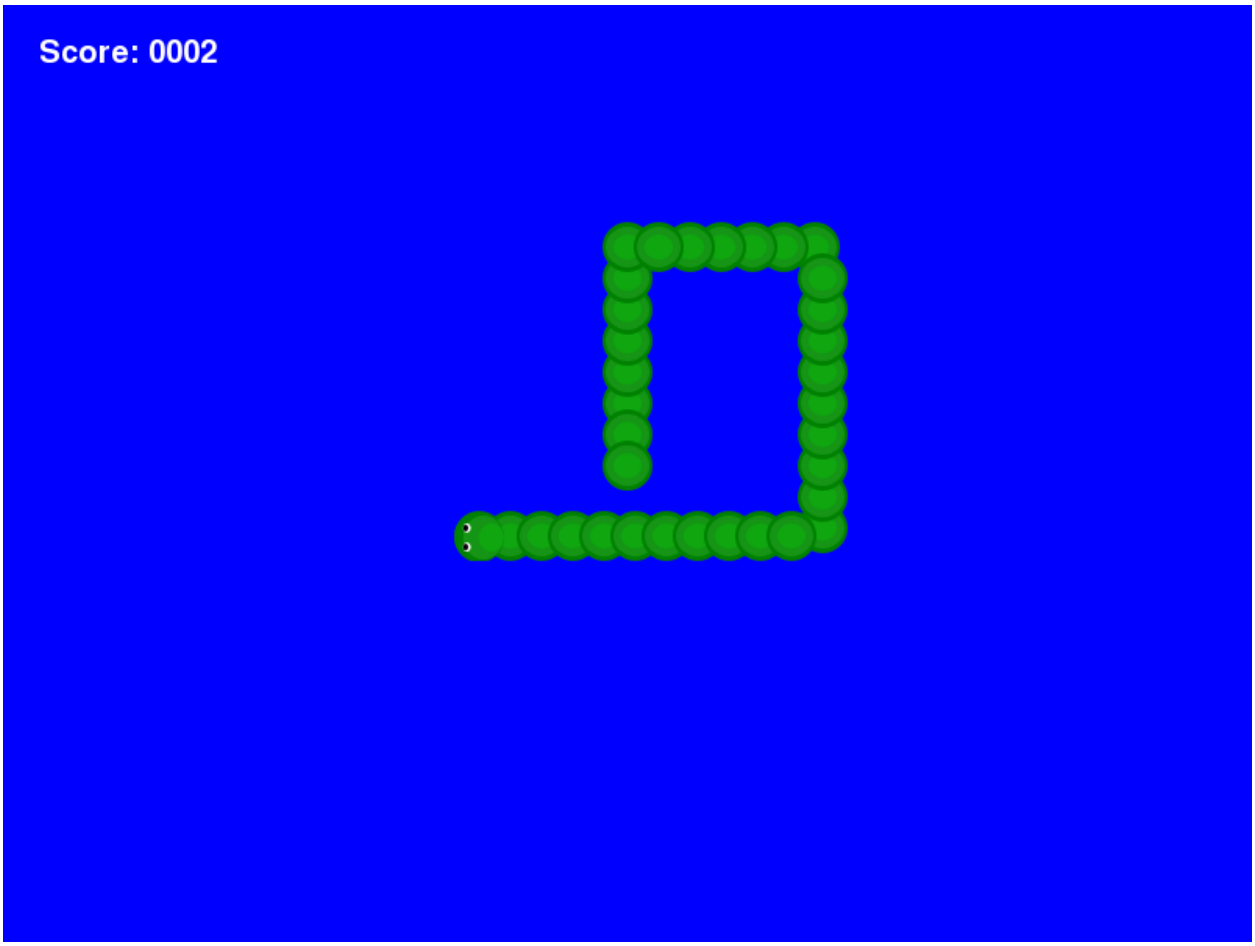
1.2 Tutorial 2: Snake Part 2

Contents

This tutorial continues on from *Tutorial 1: Snake Part 1*.

The basic game is in place and we will now add sprites, sound, fonts, animation and events.

1.2.1 Adding Sprites



The game is working ok but we need to add some sprites to start to make it look like a real game. We will add a sprite for the head and one for the body.

To use sprites with serge you first need to register them in the sprite registry. You can create simple sprites and animated sprites from single files (sprite sheets) or multiple files. For the moment let's just stick with simple sprites.

To register a sprite you use code like this.

```
import serge.visual
serge.visual.Sprites.setPath('graphics')
serge.visual.Sprites.registerItem('head', 'head.png')
serge.visual.Sprites.registerItem('tail', 'tail.png')
```

It is best to put all sprites in a separate folder. You then use *setPath* to point serge at the folder. Then you make repeated calls to *registerItem* to register each sprite. You give the sprite a name and the file that you want to use.

Once you have registered a sprite you then use it for an actor's visual by calling the *setSpriteName* method of the actor. For example.

```
snake.setSpriteName('head')
```

Now the head sprite will be used.

Let's add that to our code now. You can download the graphics here (zipfile).

```
import pygame

import serge.visual
import serge.engine
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils
import serge.blocks.directions
import serge.blocks.behaviours
import serge.blocks.actors

class Snake(serge.actor.CompositeActor):
    """Represents the snake"""

    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
        self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
        self.setSpriteName('head')
        self.setLayerName('middle')
        self.current_direction = serge.blocks.directions.N
        self.is_dying = False

    def addedToWorld(self, world):
        """The snake was added to the world"""
        super(Snake, self).addedToWorld(world)
        #
        self.keyboard = serge.engine.CurrentEngine().getKeyboard()
        self.manager = serge.blocks.behaviours.BehaviourManager('manager', 'behaviour-manager')
        world.addActor(self.manager)
        #
        # Text to display when the game is over
        self.restart_text = serge.blocks.utils.addVisualActorToWorld(world, 'text', 'restart',
            serge.visual.Text('Game Over - Press ENTER to restart', (255, 255, 255), font_size=20),
            layer_name='front',
            center_position=(400, 300))
        self.restart_text.visible = False
        #
        # A background for the game
        self.bg = serge.blocks.utils.addVisualActorToWorld(world, 'bg', 'bg',
            serge.blocks.visualblocks.Rectangle((800, 600), (0,0,255)),
            layer_name='back',
            center_position=(400, 300))
        #
        # Text to show the score
        self.score = serge.blocks.utils.addActorToWorld(world,
            serge.blocks.actors.NumericText('text', 'score', 'Score: %04d',
                (255, 255, 255), font_size=20, value=0, align='left'),
            layer_name='front',
            center_position=(80, 30))

    def updateActor(self, interval, world):
        """Update the snake"""
        super(Snake, self).updateActor(interval, world)
        #
        # Quit if requested
        if self.keyboard.isClicked(pygame.K_ESCAPE):
            serge.engine.CurrentEngine().stop()
```

```
59          #
60          # Move the head
61          if self.keyboard.isClicked(pygame.K_LEFT):
62              rotation = +90
63          elif self.keyboard.isClicked(pygame.K_RIGHT):
64              rotation = -90
65          else:
66              rotation = 0
67          #
68          # Change direction
69          if rotation:
70              current_angle = serge.blocks.directions.getAngleFromCardinal(self.current_direction)
71              self.current_direction = serge.blocks.directions.getCardinalFromAngle(current_angle+rotat
72              self.visual.setAngle(current_angle+rotation)
73          #
74          # Move
75          if not self.is_dying:
76              offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
77              self.move(*offset)
78              #
79              # Add a new segment if needed
80              if not self.getChildren() or self.getDistanceFrom(self.getChildren()[-1]) > 16:
81                  self.addSegment()
82              #
83              # Check if we hit the body
84              if self.hitBody() or self.offScreen():
85                  self.initiateDeathAnimation()
86              #
87              # Increase score
88              self.score.value += interval/1000.0
89          elif self.animation.isComplete():
90              if self.keyboard.isClicked(pygame.K_KP_ENTER) or self.keyboard.isClicked(pygame.K_RETURN)
91                  self.restartGame()
92
93      def addSegment(self):
94          """Add a new body segment"""
95          segment = serge.actor.Actor('segment')
96          segment.visual = serge.blocks.visualblocks.Circle(16, (0,200,0))
97          segment.setSpriteName('tail')
98          segment.setLayerName('middle')
99          segment.moveTo(self.x, self.y)
100         self.addChild(segment)
101
102     def hitBody(self):
103         """Return True if the head has hit the body
104
105         Look to see if we overlap with any body segment except the last
106         (we are allowed to overlap the last since we just put it down)
107
108         """
109         for segment in self.getChildren()[:-1]:
110             if self.getDistanceFrom(segment) < 16:
111                 return True
112         return False
113
114     def offScreen(self):
115         """Return True if we are off the screen"""
116         return self.x < 0 or self.x > 800 or self.y < 0 or self.y > 600
```

```
117
118     def initiateDeathAnimation(self):
119         """Begin showing the death of the snake"""
120         self.log.info('Snake died!')
121         self.animation = self.manager.assignBehaviour(self,
122             serge.blocks.behaviours.TimedCallback(1000/len(self.getChildren()), self.removeTail), 'de
123         self.is_dying = True
124         for segment in self.getChildren():
125             segment.setSpriteName('red-tail')
126         self.setSpriteName('red-head')
127
128     def removeTail(self, world, actor, interval):
129         """Remove part of the tail"""
130         self.log.debug('Removing part of the tail')
131         if self.getChildren():
132             self.removeChild(self.getChildren()[0])
133         else:
134             self.animation.markComplete()
135             self.restart_text.visible = True
136
137     def restartGame(self):
138         """Restart the game"""
139         self.is_dying = False
140         self.restart_text.visible = False
141         self.setSpriteName('head')
142         self.current_direction = serge.blocks.directions.N
143         self.score.value = 0
144         self.moveTo(400, 300)
145
146 # Create the engine
147 engine = serge.blocks.utils.getSimpleSetup(800, 600)
148 world = engine.getWorld('lab')
149
150 # Register sprites
151 serge.visual.Sprites.setPath('graphics')
152 serge.visual.Sprites.registerItem('head', 'head.png')
153 serge.visual.Sprites.registerItem('tail', 'tail.png')
154 serge.visual.Sprites.registerItem('red-head', 'red-head.png')
155 serge.visual.Sprites.registerItem('red-tail', 'red-tail.png')
156
157 # Create the snake
158 snake = Snake()
159 world.addActor(snake)
160 snake.moveTo(400, 300)
161
162 # Run the game
163 engine.run(60)
```

We didn't have to make too many changes to get this to work. One thing we did do was to create two sprites to represent the green and red states of the snake. We cannot just change the colour like we did for the circle. You could create a multi-celled sprite to do this but it is just as easy to use multiple sprites.

The other thing to notice is that we didn't have to make sprites for all the different orientations of the head. We can just use the *setAngle* method of the sprite (the actor's *visual*) to rotate the sprite in the right way.

1.2.2 Adding Sound

Sound, like sprites, must be registered before you use it. The process is very similar as it uses the same underlying *registry* approach as sprites.

To play a sound you use the following code,

```
import serge.sound
serge.sound.Sounds.setPath('sounds')
serge.sound.Sounds.registerItem('new-body', 'bloop.wav')
serge.sound.Sounds.play('new-body')
#
# Or...
my_sound = serge.sound.Sounds.getItem('new-body')
my_sound.play()
```

In our game we are going to make a sound whenever a new body piece is added and then a different one when the snake dies. Since we have the death animation, the death sound is quite long. We use the *fadeout* method of the sound to make sure that the death sound ends at approximately the same time as the on-screen animation.

```
import pygame

import serge.visual
import serge.sound
import serge.engine
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils
import serge.blocks.directions
import serge.blocks.behaviours
import serge.blocks.actors

class Snake(serge.actor.CompositeActor):
    """Represents the snake"""

    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
        self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
        self.setSpriteName('head')
        self.setLayerName('middle')
        self.current_direction = serge.blocks.directions.N
        self.is_dying = False

    def addedToWorld(self, world):
        """The snake was added to the world"""
        super(Snake, self).addedToWorld(world)
        #
        self.keyboard = serge.engine.CurrentEngine().getKeyboard()
        self.manager = serge.blocks.behaviours.BehaviourManager('manager', 'behaviour-manager')
        world.addActor(self.manager)
        #
        # Text to display when the game is over
        self.restart_text = serge.blocks.utils.addVisualActorToWorld(world, 'text', 'restart',
            serge.visual.Text('Game Over - Press ENTER to restart', (255, 255, 255), font_size=20),
            layer_name='front',
            center_position=(400, 300))
        self.restart_text.visible = False
        #
```

```
        # A background for the game
        self.bg = serge.blocks.utils.addVisualActorToWorld(world, 'bg', 'bg',
            serge.blocks.visualblocks.Rectangle((800, 600), (0,0,255)),
            layer_name='back',
            center_position=(400, 300))
        #
        # Text to show the score
        self.score = serge.blocks.utils.addActorToWorld(world,
            serge.blocks.actors.NumericText('text', 'score', 'Score: %04d',
                (255, 255, 255), font_size=20, value=0, align='left'),
            layer_name='front',
            center_position=(80, 30))

    def updateActor(self, interval, world):
        """Update the snake"""
        super(Snake, self).updateActor(interval, world)
        #
        # Quit if requested
        if self.keyboard.isClicked(pygame.K_ESCAPE):
            serge.engine.CurrentEngine().stop()
        #
        # Move the head
        if self.keyboard.isClicked(pygame.K_LEFT):
            rotation = +90
        elif self.keyboard.isClicked(pygame.K_RIGHT):
            rotation = -90
        else:
            rotation = 0
        #
        # Change direction
        if rotation:
            current_angle = serge.blocks.directions.getAngleFromCardinal(self.current_direction)
            self.current_direction = serge.blocks.directions.getCardinalFromAngle(current_angle+rotat
            self.visual.setAngle(current_angle+rotation)
        #
        # Move
        if not self.is_dying:
            offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
            self.move(*offset)
            #
            # Add a new segment if needed
            if not self.getChildren() or self.getDistanceFrom(self.getChildren()[-1]) > 16:
                self.addSegment()
            #
            # Check if we hit the body
            if self.hitBody() or self.offScreen():
                self.initiateDeathAnimation()
            #
            # Increase score
            self.score.value += interval/1000.0
        elif self.animation.isComplete():
            if self.keyboard.isClicked(pygame.K_KP_ENTER) or self.keyboard.isClicked(pygame.K_RETURN)
                self.restartGame()

    def addSegment(self):
        """Add a new body segment"""
        segment = serge.actor.Actor('segment')
        segment.visual = serge.blocks.visualblocks.Circle(16, (0,200,0))
```

```
98          segment.setSpriteName('tail')
99          segment.setLayerName('middle')
100         segment.moveTo(self.x, self.y)
101         self.addChild(segment)
102         serge.sound.Sounds.play('new-body')
103
104     def hitBody(self):
105         """Return True if the head has hit the body
106
107         Look to see if we overlap with any body segment except the last
108         (we are allowed to overlap the last since we just put it down)
109
110         """
111         for segment in self.getChildren()[:-1]:
112             if self.getDistanceFrom(segment) < 16:
113                 return True
114         return False
115
116     def offScreen(self):
117         """Return True if we are off the screen"""
118         return self.x < 0 or self.x > 800 or self.y < 0 or self.y > 600
119
120     def initiateDeathAnimation(self):
121         """Begin showing the death of the snake"""
122         self.log.info('Snake died!')
123         self.animation = self.manager.assignBehaviour(self,
124             serge.blocks.behaviours.TimedCallback(1000/len(self.getChildren()), self.removeTail), 'de
125         self.is_dying = True
126         for segment in self.getChildren():
127             segment.setSpriteName('red-tail')
128         self.setSpriteName('red-head')
129         serge.sound.Sounds.play('snake-death')
130
131     def removeTail(self, world, actor, interval):
132         """Remove part of the tail"""
133         self.log.debug('Removing part of the tail')
134         if self.getChildren():
135             self.removeChild(self.getChildren()[0])
136         else:
137             self.animation.markComplete()
138             self.restart_text.visible = True
139             serge.sound.Sounds.getItem('snake-death').fadeout(500)
140
141     def restartGame(self):
142         """Restart the game"""
143         self.is_dying = False
144         self.restart_text.visible = False
145         self.setSpriteName('head')
146         self.current_direction = serge.blocks.directions.N
147         self.score.value = 0
148         self.moveTo(400, 300)
149
150 # Create the engine
151 engine = serge.blocks.utils.getSimpleSetup(800, 600)
152 world = engine.getWorld('lab')
153
154 # Register sprites
155 serge.visual.Sprites.setPath('graphics')
```

```
156 serge.visual.Sprites.registerItem('head', 'head.png')
157 serge.visual.Sprites.registerItem('tail', 'tail.png')
158 serge.visual.Sprites.registerItem('red-head', 'red-head.png')
159 serge.visual.Sprites.registerItem('red-tail', 'red-tail.png')
160
161 # Register sounds
162 serge.sound.Sounds.setPath('sounds')
163 serge.sound.Sounds.registerItem('new-body', 'bloop.wav')
164 serge.sound.Sounds.registerItem('snake-death', 'death.wav')
165
166 # Create the snake
167 snake = Snake()
168 world.addActor(snake)
169 snake.moveTo(400, 300)
170
171 # Run the game
172 engine.run(60)
```

1.2.3 Adding Fonts

The default fonts in pygame are good but it adds a nice touch to include a custom font. The process for using fonts is very similar to sound and graphics. You need to register the font location, register a font and then you can refer to it subsequently by the registered name.

```
import serge.visual
serge.visual.Fonts.setPath('fonts')
serge.visual.Fonts.registerItem('DEFAULT', 'MedievalSharp.ttf')
serge.visual.Fonts.registerItem('scores', 'PressStart2P.ttf')
```

You for fonts there is also a special name, *DEFAULT*. If you register a font with this name then this will be the one used by default for all text.

We are using two fonts here, one for the main text and one for the scores. You probably don't need to do this in such a simple game but it allows us to see the difference between using the default font and a named font. All classes involving text take some kind of *font_name* parameter. If you do not pass anything then the default font is used. Alternatively you pass the name of a registered font and it will use that one.

Note that in the updated game we had to move the score text over a bit as the chosen font is larger than the default.

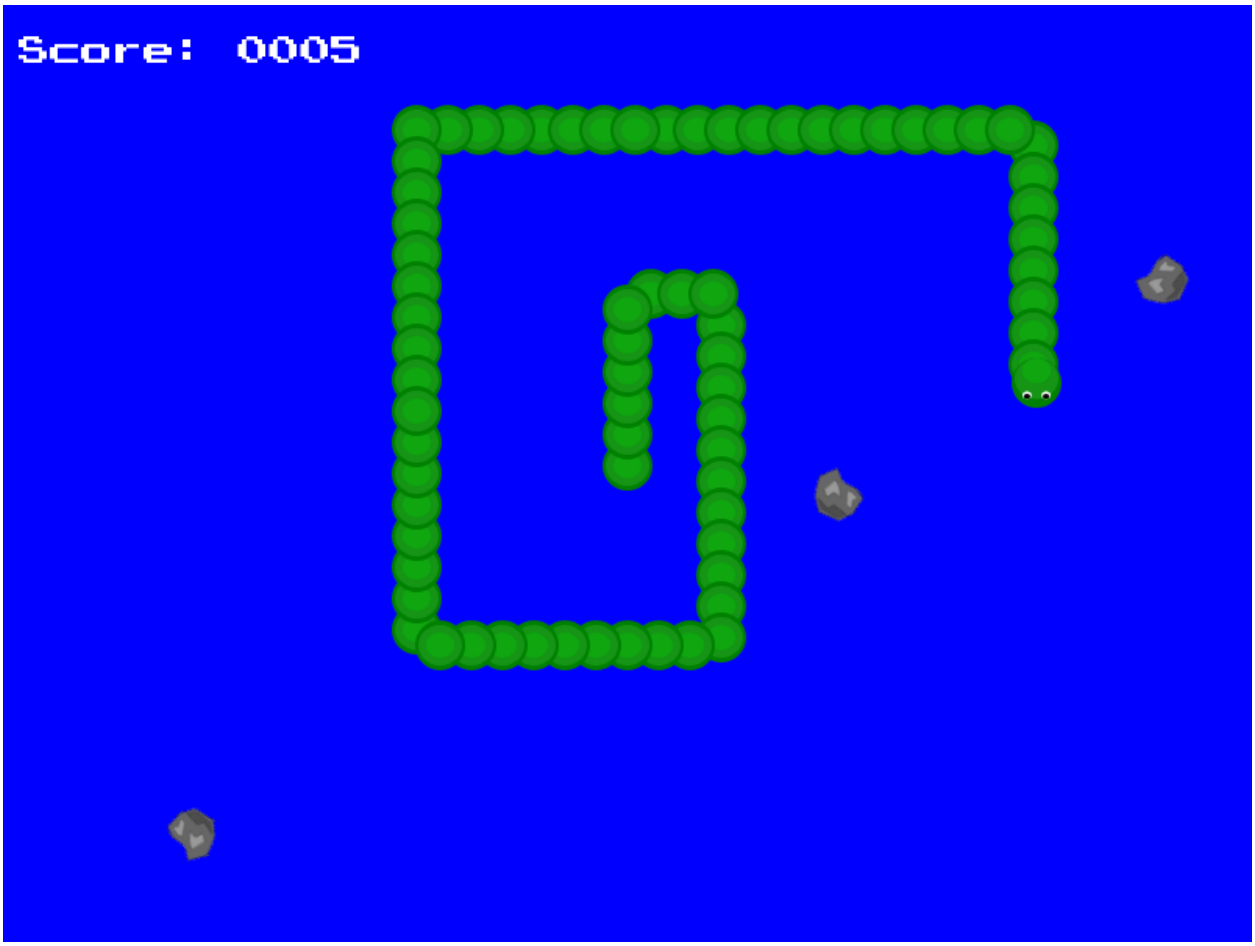
```
import pygame

import serge.visual
import serge.sound
import serge.engine
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils
import serge.blocks.directions
import serge.blocks.behaviours
import serge.blocks.actors

class Snake(serge.actor.CompositeActor):
    """Represents the snake"""

    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
        self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
        self.setSpriteName('head')
        self.setLayerName('middle')
        self.current_direction = serge.blocks.directions.N
        self.is_dying = False

    def addedToWorld(self, world):
        """The snake was added to the world"""
        super(Snake, self).addedToWorld(world)
        #
        self.keyboard = serge.engine.CurrentEngine().getKeyboard()
        self.manager = serge.blocks.behaviours.BehaviourManager('manager', 'behaviour-manager')
        world.addActor(self.manager)
        #
        # Text to display when the game is over
        self.restart_text = serge.blocks.utils.addVisualActorToWorld(world, 'text', 'restart',
            serge.visual.Text('Game Over - Press ENTER to restart', (255, 255, 255), font_size=20),
            layer_name='front',
            center_position=(400, 300))
        self.restart_text.visible = False
        #
        # A background for the game
        self.bg = serge.blocks.utils.addVisualActorToWorld(world, 'bg', 'bg',
            serge.blocks.visualblocks.Rectangle((800, 600), (0,0,255)),
            layer_name='back',
            center_position=(400, 300))
```

```
45          #
46          # Text to show the score
47          self.score = serge.blocks.utils.addActorToWorld(world,
48              serge.blocks.actors.NumericText('text', 'score', 'Score: %04d',
49                  (255, 255, 255), font_size=20, font_name='scores', value=0, align='left'),
50              layer_name='front',
51              center_position=(120, 30))
52
53      def updateActor(self, interval, world):
54          """Update the snake"""
55          super(Snake, self).updateActor(interval, world)
56          #
57          # Quit if requested
58          if self.keyboard.isClicked(pygame.K_ESCAPE):
59              serge.engine.CurrentEngine().stop()
60          #
61          # Move the head
62          if self.keyboard.isClicked(pygame.K_LEFT):
63              rotation = +90
64          elif self.keyboard.isClicked(pygame.K_RIGHT):
65              rotation = -90
66          else:
67              rotation = 0
68          #
69          # Change direction
70          if rotation:
71              current_angle = serge.blocks.directions.getAngleFromCardinal(self.current_direction)
72              self.current_direction = serge.blocks.directions.getCardinalFromAngle(current_angle+rotat
73              self.visual.setAngle(current_angle+rotation)
74          #
75          # Move
76          if not self.is_dying:
77              offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
78              self.move(*offset)
79              #
80              # Add a new segment if needed
81              if not self.getChildren() or self.getDistanceFrom(self.getChildren()[-1]) > 16:
82                  self.addSegment()
83              #
84              # Check if we hit the body
85              if self.hitBody() or self.offScreen():
86                  self.initiateDeathAnimation()
87              #
88              # Increase score
89              self.score.value += interval/1000.0
90          elif self.animation.isComplete():
91              if self.keyboard.isClicked(pygame.K_KP_ENTER) or self.keyboard.isClicked(pygame.K_RETURN)
92                  self.restartGame()
93
94      def addSegment(self):
95          """Add a new body segment"""
96          segment = serge.actor.Actor('segment')
97          segment.visual = serge.blocks.visualblocks.Circle(16, (0,200,0))
98          segment.setSpriteName('tail')
99          segment.setLayerName('middle')
100         segment.moveTo(self.x, self.y)
101         self.addChild(segment)
102         serge.sound.Sounds.play('new-body')
```

```
103
104     def hitBody(self):
105         """Return True if the head has hit the body
106
107         Look to see if we overlap with any body segment except the last
108         (we are allowed to overlap the last since we just put it down)
109
110         """
111         for segment in self.getChildren()[:-1]:
112             if self.getDistanceFrom(segment) < 16:
113                 return True
114         return False
115
116     def offScreen(self):
117         """Return True if we are off the screen"""
118         return self.x < 0 or self.x > 800 or self.y < 0 or self.y > 600
119
120     def initiateDeathAnimation(self):
121         """Begin showing the death of the snake"""
122         self.log.info('Snake died!')
123         self.animation = self.manager.assignBehaviour(self,
124             serge.blocks.behaviours.TimedCallback(1000/len(self.getChildren()), self.removeTail), 'de
125         self.is_dying = True
126         for segment in self.getChildren():
127             segment.setSpriteName('red-tail')
128         self.setSpriteName('red-head')
129         serge.sound.Sounds.play('snake-death')
130
131     def removeTail(self, world, actor, interval):
132         """Remove part of the tail"""
133         self.log.debug('Removing part of the tail')
134         if self.getChildren():
135             self.removeChild(self.getChildren()[0])
136         else:
137             self.animation.markComplete()
138             self.restart_text.visible = True
139             serge.sound.Sounds.getItem('snake-death').fadeout(500)
140
141     def restartGame(self):
142         """Restart the game"""
143         self.is_dying = False
144         self.restart_text.visible = False
145         self.setSpriteName('head')
146         self.current_direction = serge.blocks.directions.N
147         self.score.value = 0
148         self.moveTo(400, 300)
149
150 # Create the engine
151 engine = serge.blocks.utils.getSimpleSetup(800, 600)
152 world = engine.getWorld('lab')
153
154 # Register sprites
155 serge.visual.Sprites.setPath('graphics')
156 serge.visual.Sprites.registerItem('head', 'head.png')
157 serge.visual.Sprites.registerItem('tail', 'tail.png')
158 serge.visual.Sprites.registerItem('red-head', 'red-head.png')
159 serge.visual.Sprites.registerItem('red-tail', 'red-tail.png')
160
```

```
161 # Register sounds
162 serge.sound.Sounds.setPath('sounds')
163 serge.sound.Sounds.registerItem('new-body', 'bloop.wav')
164 serge.sound.Sounds.registerItem('snake-death', 'death.wav')
165
166 # Register fonts
167 serge.visual.Fonts.setPath('fonts')
168 serge.visual.Fonts.registerItem('DEFAULT', 'MedievalSharp.ttf')
169 serge.visual.Fonts.registerItem('scores', 'PressStart2P.ttf')
170
171 # Create the snake
172 snake = Snake()
173 world.addActor(snake)
174 snake.moveTo(400, 300)
175
176 # Run the game
177 engine.run(60)
```

1.2.4 Additional Gameplay elements



Before exploring more of the game engine we need to add some more gameplay elements.

Let's add a number of rocks to the screen. If the snake hits a rock then it is going to die. But later we will allow the player to click on the rocks to blow them up.

First we need to add a rock graphic and then add some code to add it to the screen. We register the rock graphic as before, with:

```
serge.visual.Sprites.regsiterItem('rock', 'rock.png')
```

Then we will randomly add a rock to the screen every so often in the snakes *updateActor* method. We also need to check if the snake has hit a rock. We do this in the same method.

When we add a rock we use the line:

```
rock = serge.actor.Actor('rock')
```

The text *'rock'* here is the actor's *tag*. Tags are very useful and can be used to locate groups of actors in the world. In this case we are going to use it to later find out all the rocks that we have added without having to manually keep track.

Every actor has a tag and optionally can have a name. You can also find actors by names but names are assumed (but not forced) to be unique.

```
rock = serge.actor.Actor('rock', 'rock-63')
```

The new code is as follows.

```
import random
import pygame

import serge.visual
import serge.sound
import serge.engine
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils
import serge.blocks.directions
import serge.blocks.behaviours
import serge.blocks.actors

class Snake(serge.actor.CompositeActor):
    """Represents the snake"""

    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
        self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
        self.setSpriteName('head')
        self.setLayerName('middle')
        self.current_direction = serge.blocks.directions.N
        self.is_dying = False

    def addedToWorld(self, world):
        """The snake was added to the world"""
        super(Snake, self).addedToWorld(world)
        #
        self.keyboard = serge.engine.CurrentEngine().getKeyboard()
        self.manager = serge.blocks.behaviours.BehaviourManager('manager', 'behaviour-manager')
        self.world = world
        world.addActor(self.manager)
        #
        # Text to display when the game is over
        self.restart_text = serge.blocks.utils.addVisualActorToWorld(world, 'text', 'restart',
            serge.visual.Text('Game Over - Press ENTER to restart', (255, 255, 255), font_size=20),
            layer_name='front',
```

```
            center_position=(400, 300))
        self.restart_text.visible = False
        #
        # A background for the game
        self.bg = serge.blocks.utils.addVisualActorToWorld(world, 'bg', 'bg',
            serge.blocks.visualblocks.Rectangle((800, 600), (0,0,255)),
            layer_name='back',
            center_position=(400, 300))
        #
        # Text to show the score
        self.score = serge.blocks.utils.addActorToWorld(world,
            serge.blocks.actors.NumericText('text', 'score', 'Score: %04d',
                (255, 255, 255), font_size=20, font_name='scores', value=0, align='left'),
            layer_name='front',
            center_position=(120, 30))

    def updateActor(self, interval, world):
        """Update the snake"""
        super(Snake, self).updateActor(interval, world)
        #
        # Quit if requested
        if self.keyboard.isClicked(pygame.K_ESCAPE):
            serge.engine.CurrentEngine().stop()
        #
        # Move the head
        if self.keyboard.isClicked(pygame.K_LEFT):
            rotation = +90
        elif self.keyboard.isClicked(pygame.K_RIGHT):
            rotation = -90
        else:
            rotation = 0
        #
        # Change direction
        if rotation:
            current_angle = serge.blocks.directions.getAngleFromCardinal(self.current_direction)
            self.current_direction = serge.blocks.directions.getCardinalFromAngle(current_angle+rotat
            self.visual.setAngle(current_angle+rotation)
        #
        # Move
        if not self.is_dying:
            offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
            self.move(*offset)
            #
            # Adding random rocks
            if random.random() < 0.01:
                self.addRock()
            #
            # Add a new segment if needed
            if not self.getChildren() or self.getDistanceFrom(self.getChildren()[-1]) > 16:
                self.addSegment()
            #
            # Check if we hit the body
            if self.hitBody() or self.offScreen() or self.hitRock():
                self.initiateDeathAnimation()
            #
            # Increase score
            self.score.value += interval/1000.0
        elif self.animation.isComplete():
```

```
97              if self.keyboard.isClicked(pygame.K_KP_ENTER) or self.keyboard.isClicked(pygame.K_RETURN)
98                  self.restartGame()
99
100     def addSegment(self):
101         """Add a new body segment"""
102         segment = serge.actor.Actor('segment')
103         segment.setSpriteName('tail')
104         segment.setLayerName('middle')
105         segment.moveTo(self.x, self.y)
106         self.addChild(segment)
107         serge.sound.Sounds.play('new-body')
108
109     def addRock(self):
110         """Add a rock to the screen"""
111         position = (random.randrange(0, 800), random.randrange(0, 600))
112         rock = serge.actor.Actor('rock')
113         rock.setSpriteName('rock')
114         rock.setLayerName('middle')
115         rock.moveTo(*position)
116         rock.setAngle(random.randrange(0, 360))
117         self.world.addActor(rock)
118
119     def hitBody(self):
120         """Return True if the head has hit the body
121
122         Look to see if we overlap with any body segment except the last
123         (we are allowed to overlap the last since we just put it down)
124
125         """
126         for segment in self.getChildren()[:-1]:
127             if self.getDistanceFrom(segment) < 16:
128                 return True
129         return False
130
131     def hitRock(self):
132         """Return True if we hit a rock"""
133         for rock in self.world.findActorsByTag('rock'):
134             if self.getDistanceFrom(rock) < 16:
135                 return True
136         else:
137             return False
138
139     def offScreen(self):
140         """Return True if we are off the screen"""
141         return self.x < 0 or self.x > 800 or self.y < 0 or self.y > 600
142
143     def initiateDeathAnimation(self):
144         """Begin showing the death of the snake"""
145         self.log.info('Snake died!')
146         self.animation = self.manager.assignBehaviour(self,
147             serge.blocks.behaviours.TimedCallback(1000/len(self.getChildren()), self.removeTail), 'de
148         self.is_dying = True
149         for segment in self.getChildren():
150             segment.setSpriteName('red-tail')
151         self.setSpriteName('red-head')
152         serge.sound.Sounds.play('snake-death')
153
154     def removeTail(self, world, actor, interval):
```

```
155         """Remove part of the tail"""
156         self.log.debug('Removing part of the tail')
157         if self.getChildren():
158             self.removeChild(self.getChildren()[0])
159         else:
160             self.animation.markComplete()
161             self.restart_text.visible = True
162             serge.sound.Sounds.getItem('snake-death').fadeout(500)
163
164     def restartGame(self):
165         """Restart the game"""
166         self.is_dying = False
167         self.restart_text.visible = False
168         self.setSpriteName('head')
169         self.current_direction = serge.blocks.directions.N
170         self.score.value = 0
171         self.moveTo(400, 300)
172         self.world.clearActorsWithTags(['rock'])
173
174
175 # Create the engine
176 engine = serge.blocks.utils.getSimpleSetup(800, 600)
177 world = engine.getWorld('lab')
178
179 # Register sprites
180 serge.visual.Sprites.setPath('graphics')
181 serge.visual.Sprites.registerItem('head', 'head.png')
182 serge.visual.Sprites.registerItem('tail', 'tail.png')
183 serge.visual.Sprites.registerItem('red-head', 'red-head.png')
184 serge.visual.Sprites.registerItem('red-tail', 'red-tail.png')
185 serge.visual.Sprites.registerItem('rock', 'rock.png')
186
187 # Register sounds
188 serge.sound.Sounds.setPath('sounds')
189 serge.sound.Sounds.registerItem('new-body', 'bloop.wav')
190 serge.sound.Sounds.registerItem('snake-death', 'death.wav')
191
192 # Register fonts
193 serge.visual.Fonts.setPath('fonts')
194 serge.visual.Fonts.registerItem('DEFAULT', 'MedievalSharp.ttf')
195 serge.visual.Fonts.registerItem('scores', 'PressStart2P.ttf')
196
197 # Create the snake
198 snake = Snake()
199 world.addActor(snake)
200 snake.moveTo(400, 300)
201
202 # Run the game
203 engine.run(60)
```

Some things to note here:

- We store the *world* object in the *addActorToWorld* as we are going to use this a lot later. This is quite a common requirement and you will find that you often need to do this.
- We use the *world.findActorsByTags* method to locate all the rocks. We didn't need to use this for finding the tail segments because we stored these as children. We could have used a similar approach but it is often best not to store lists of actors but just to find them in the world by their tags. If you keep lists of actors hanging around then you need to keep them up to date when actors get added and removed from the world and the world does

this anyway so it often isn't worth the minor speed improvement to keep the lists yourself.

- We again use the tags to help remove all the rocks from the world when we are restarting (*world.clearActorsWithTags*)

1.2.5 Events

Ok, so now we have rocks being added to the screen and we have to dodge them. Let's allow the user to blow them up by clicking on them.

To do this we can use the event and notification system. You can link many game events to your own functions to easily react when something happens. In this case we want to do something when the user clicks the mouse on a rock.

```
rock.linkEvent(serge.events.E_LEFT_CLICK, self.destroyRock, rock)
```

Our method *destroyRock*, which we havent' written yet, will be called whenever the user clicks on a rock. There are many different kinds of events. You can look at these in the `serge.events` module. You can be notified when actors are added or removed from the world or when worlds are activated or deactivated. You can even create your own events and use these to trigger actions like starting a new game.

We pass rock as an additional parameter to *linkEvent* because the event callback (*destroyRock*) will be called as *destroyRock(obj, arg)* with *obj* being the object involved in the event and *arg* being the final parameter in the *linkEvent* call. In the case of *E_LEFT_CLICK* the *obj* parameter is the *mouse* object and we really want to know which rock was clicked on so we pass this as the *arg* parameter.

For some events you do not need this and you can omit the *arg* parameter.

The new code is below. Try it out and try clicking on the rocks as they appear.

```
import random
import pygame

import serge.visual
import serge.sound
import serge.engine
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils
import serge.blocks.directions
import serge.blocks.behaviours
import serge.blocks.actors

class Snake(serge.actor.CompositeActor):
    """Represents the snake"""

    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
        self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
        self.setSpriteName('head')
        self.setLayerName('middle')
        self.current_direction = serge.blocks.directions.N
        self.is_dying = False

    def addedToWorld(self, world):
        """The snake was added to the world"""
        super(Snake, self).addedToWorld(world)
        #
        self.keyboard = serge.engine.CurrentEngine().getKeyboard()
```

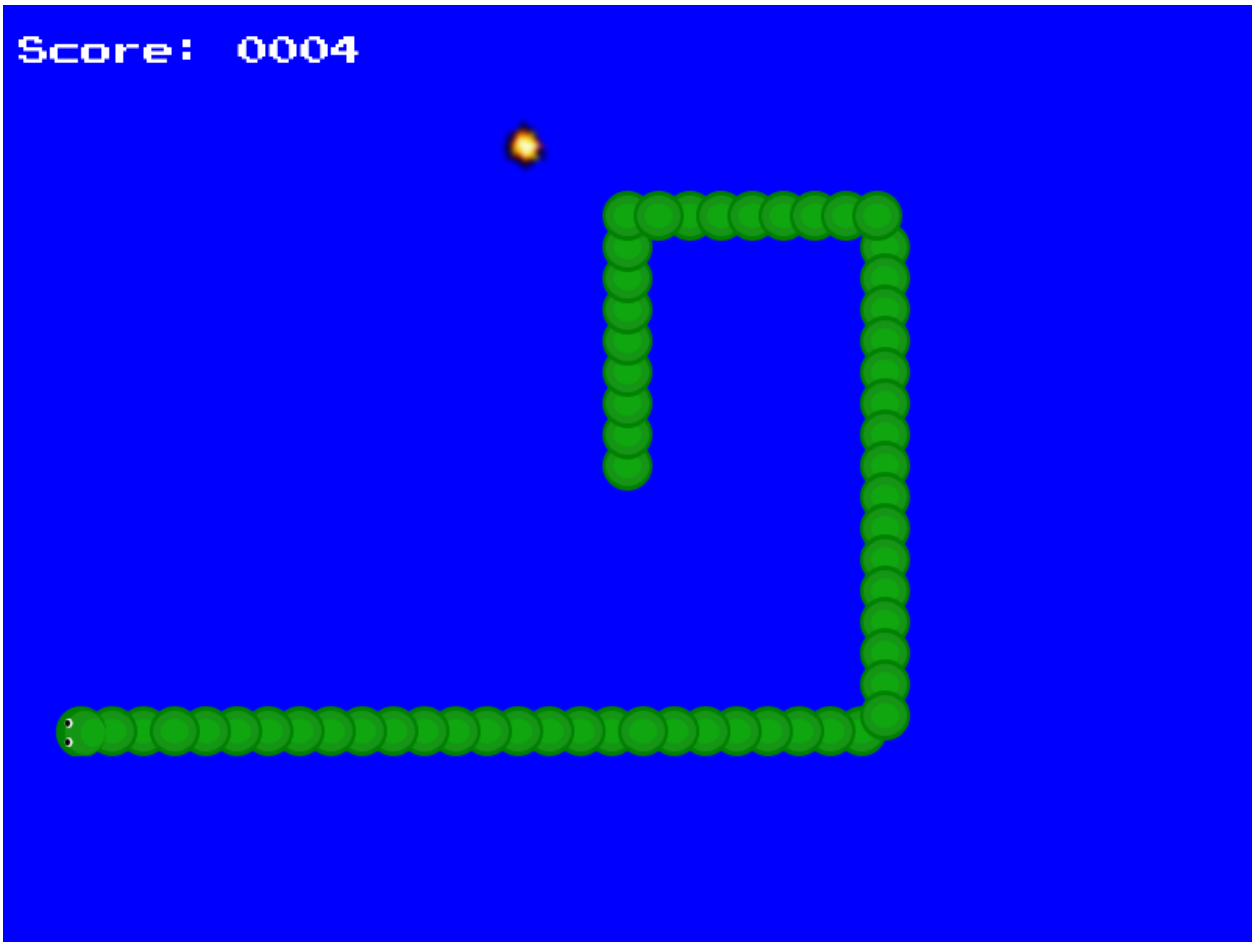
```
31          self.manager = serge.blocks.behaviours.BehaviourManager('manager', 'behaviour-manager')
32          self.world = world
33          world.addActor(self.manager)
34          #
35          # Text to display when the game is over
36          self.restart_text = serge.blocks.utils.addVisualActorToWorld(world, 'text', 'restart',
37              serge.visual.Text('Game Over - Press ENTER to restart', (255, 255, 255), font_size=20),
38              layer_name='front',
39              center_position=(400, 300))
40          self.restart_text.visible = False
41          #
42          # A background for the game
43          self.bg = serge.blocks.utils.addVisualActorToWorld(world, 'bg', 'bg',
44              serge.blocks.visualblocks.Rectangle((800, 600), (0,0,255)),
45              layer_name='back',
46              center_position=(400, 300))
47          #
48          # Text to show the score
49          self.score = serge.blocks.utils.addActorToWorld(world,
50              serge.blocks.actors.NumericText('text', 'score', 'Score: %04d',
51                  (255, 255, 255), font_size=20, font_name='scores', value=0, align='left'),
52              layer_name='front',
53              center_position=(120, 30))
54
55      def updateActor(self, interval, world):
56          """Update the snake"""
57          super(Snake, self).updateActor(interval, world)
58          #
59          # Quit if requested
60          if self.keyboard.isClicked(pygame.K_ESCAPE):
61              serge.engine.CurrentEngine().stop()
62          #
63          # Move the head
64          if self.keyboard.isClicked(pygame.K_LEFT):
65              rotation = +90
66          elif self.keyboard.isClicked(pygame.K_RIGHT):
67              rotation = -90
68          else:
69              rotation = 0
70          #
71          # Change direction
72          if rotation:
73              current_angle = serge.blocks.directions.getAngleFromCardinal(self.current_direction)
74              self.current_direction = serge.blocks.directions.getCardinalFromAngle(current_angle+rotat
75              self.visual.setAngle(current_angle+rotation)
76          #
77          # Move
78          if not self.is_dying:
79              offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
80              self.move(*offset)
81              #
82              # Adding random rocks
83              if random.random() < 0.01:
84                  self.addRock()
85              #
86              # Add a new segment if needed
87              if not self.getChildren() or self.getDistanceFrom(self.getChildren()[-1]) > 16:
88                  self.addSegment()
```

```
89              #
90              # Check if we hit the body
91              if self.hitBody() or self.offScreen() or self.hitRock():
92                  self.initiateDeathAnimation()
93              #
94              # Increase score
95              self.score.value += interval/1000.0
96          elif self.animation.isComplete():
97              if self.keyboard.isClicked(pygame.K_KP_ENTER) or self.keyboard.isClicked(pygame.K_RETURN)
98                  self.restartGame()
99
100     def addSegment(self):
101         """Add a new body segment"""
102         segment = serge.actor.Actor('segment')
103         segment.setSpriteName('tail')
104         segment.setLayerName('middle')
105         segment.moveTo(self.x, self.y)
106         self.addChild(segment)
107         serge.sound.Sounds.play('new-body')
108
109     def addRock(self):
110         """Add a rock to the screen"""
111         position = (random.randrange(0, 800), random.randrange(0, 600))
112         rock = serge.actor.Actor('rock')
113         rock.setSpriteName('rock')
114         rock.setLayerName('middle')
115         rock.moveTo(*position)
116         rock.setAngle(random.randrange(0, 360))
117         rock.linkEvent(serge.events.E_LEFT_CLICK, self.destroyRock, rock)
118         self.world.addActor(rock)
119
120     def hitBody(self):
121         """Return True if the head has hit the body
122
123         Look to see if we overlap with any body segment except the last
124         (we are allowed to overlap the last since we just put it down)
125
126         """
127         for segment in self.getChildren()[:-1]:
128             if self.getDistanceFrom(segment) < 16:
129                 return True
130         return False
131
132     def hitRock(self):
133         """Return True if we hit a rock"""
134         for rock in self.world.findActorsByTag('rock'):
135             if self.getDistanceFrom(rock) < 16:
136                 return True
137         else:
138             return False
139
140     def offScreen(self):
141         """Return True if we are off the screen"""
142         return self.x < 0 or self.x > 800 or self.y < 0 or self.y > 600
143
144     def initiateDeathAnimation(self):
145         """Begin showing the death of the snake"""
146         self.log.info('Snake died!')
```

```
147             self.animation = self.manager.assignBehaviour(self,
148                 serge.blocks.behaviours.TimedCallback(1000/len(self.getChildren()), self.removeTail), 'de
149             self.is_dying = True
150             for segment in self.getChildren():
151                 segment.setSpriteName('red-tail')
152             self.setSpriteName('red-head')
153             serge.sound.Sounds.play('snake-death')
154 
155         def removeTail(self, world, actor, interval):
156             """Remove part of the tail"""
157             self.log.debug('Removing part of the tail')
158             if self.getChildren():
159                 self.removeChild(self.getChildren()[0])
160             else:
161                 self.animation.markComplete()
162                 self.restart_text.visible = True
163                 serge.sound.Sounds.getItem('snake-death').fadeout(500)
164 
165         def destroyRock(self, obj, rock):
166             """Destroy a rock"""
167             self.world.removeActor(rock)
168 
169         def restartGame(self):
170             """Restart the game"""
171             self.is_dying = False
172             self.restart_text.visible = False
173             self.setSpriteName('head')
174             self.current_direction = serge.blocks.directions.N
175             self.score.value = 0
176             self.moveTo(400, 300)
177             self.world.clearActorsWithTags(['rock'])
178 
179 
180 # Create the engine
181 engine = serge.blocks.utils.getSimpleSetup(800, 600)
182 world = engine.getWorld('lab')
183 
184 # Register sprites
185 serge.visual.Sprites.setPath('graphics')
186 serge.visual.Sprites.registerItem('head', 'head.png')
187 serge.visual.Sprites.registerItem('tail', 'tail.png')
188 serge.visual.Sprites.registerItem('red-head', 'red-head.png')
189 serge.visual.Sprites.registerItem('red-tail', 'red-tail.png')
190 serge.visual.Sprites.registerItem('rock', 'rock.png')
191 
192 # Register sounds
193 serge.sound.Sounds.setPath('sounds')
194 serge.sound.Sounds.registerItem('new-body', 'bloop.wav')
195 serge.sound.Sounds.registerItem('snake-death', 'death.wav')
196 
197 # Register fonts
198 serge.visual.Fonts.setPath('fonts')
199 serge.visual.Fonts.registerItem('DEFAULT', 'MedievalSharp.ttf')
200 serge.visual.Fonts.registerItem('scores', 'PressStart2P.ttf')
201 
202 # Create the snake
203 snake = Snake()
204 world.addActor(snake)
```

```
205 snake.moveTo(400, 300)
206
207 # Run the game
208 engine.run(60)
```

1.2.6 Animation



Clicking on the rock makes it go away but it doesn't really feel like it is exploding. Let's add a bit of animation to that by having an explosion occur.

Animations are just sprites that have a number of cells. Each cell of the animation will play for a certain length of time and then the animation will move on to the next cell. You can create cells of animation either from a single file or from multiple files. For this example we will use a single file where the cells of animation are arranged horizontally.

Here is our file.

To register this as a sprite we use:

```
serge.visual.Sprites.registerItem('explosion', 'explosion.png', zoom=0.25,
    w=8, framerate=10, running=True, loop=False, one_direction=True)
```

The framerate sets the number of cells that will be displayed per second. We do not want this animation to loop around and we only want it to go in one direction, ie we want it to run to the end of the animation and then stop. This particular graphic is actually quite large so we also use the *zoom* argument to scale it down a bit.

We want to add the animation to the screen whenever we destroy a rock. There is a useful *block* called an *AnimateThenDie* actor that we can use for this purpose. This is an actor that we place in the world and it will run its animation one and then be removed. This actor is ideal for explosions because we just want them to show and then go away. Look in the *destroyRock* method to see how we use this actor.

```
import random
import pygame

import serge.visual
import serge.sound
import serge.engine
import serge.actor
import serge.blocks.visualblocks
import serge.blocks.utils
import serge.blocks.directions
import serge.blocks.behaviours
import serge.blocks.actors

class Snake(serge.actor.CompositeActor):
    """Represents the snake"""

    def __init__(self):
        """Initialise the snake"""
        super(Snake, self).__init__('snake', 'snake-head')
        self.visual = serge.blocks.visualblocks.Circle(16, (0,255,0))
        self.setSpriteName('head')
        self.setLayerName('middle')
        self.current_direction = serge.blocks.directions.N
        self.is_dying = False

    def addedToWorld(self, world):
        """The snake was added to the world"""
        super(Snake, self).addedToWorld(world)
        #
        self.keyboard = serge.engine.CurrentEngine().getKeyboard()
        self.manager = serge.blocks.behaviours.BehaviourManager('manager', 'behaviour-manager')
        self.world = world
        world.addActor(self.manager)
        #
        # Text to display when the game is over
        self.restart_text = serge.blocks.utils.addVisualActorToWorld(world, 'text', 'restart',
            serge.visual.Text('Game Over - Press ENTER to restart', (255, 255, 255), font_size=20),
            layer_name='front',
            center_position=(400, 300))
        self.restart_text.visible = False
        #
        # A background for the game
        self.bg = serge.blocks.utils.addVisualActorToWorld(world, 'bg', 'bg',
            serge.blocks.visualblocks.Rectangle((800, 600), (0,0,255)),
            layer_name='back',
            center_position=(400, 300))
```

```
47          #
48          # Text to show the score
49          self.score = serge.blocks.utils.addActorToWorld(world,
50              serge.blocks.actors.NumericText('text', 'score', 'Score: %04d',
51                  (255, 255, 255), font_size=20, font_name='scores', value=0, align='left'),
52              layer_name='front',
53              center_position=(120, 30))
54
55      def updateActor(self, interval, world):
56          """Update the snake"""
57          super(Snake, self).updateActor(interval, world)
58          #
59          # Quit if requested
60          if self.keyboard.isClicked(pygame.K_ESCAPE):
61              serge.engine.CurrentEngine().stop()
62          #
63          # Move the head
64          if self.keyboard.isClicked(pygame.K_LEFT):
65              rotation = +90
66          elif self.keyboard.isClicked(pygame.K_RIGHT):
67              rotation = -90
68          else:
69              rotation = 0
70          #
71          # Change direction
72          if rotation:
73              current_angle = serge.blocks.directions.getAngleFromCardinal(self.current_direction)
74              self.current_direction = serge.blocks.directions.getCardinalFromAngle(current_angle+rotat
75              self.visual.setAngle(current_angle+rotation)
76          #
77          # Move
78          if not self.is_dying:
79              offset = 5*serge.blocks.directions.getVectorFromCardinal(self.current_direction)
80              self.move(*offset)
81              #
82              # Adding random rocks
83              if random.random() < 0.01:
84                  self.addRock()
85              #
86              # Add a new segment if needed
87              if not self.getChildren() or self.getDistanceFrom(self.getChildren()[-1]) > 16:
88                  self.addSegment()
89              #
90              # Check if we hit the body
91              if self.hitBody() or self.offScreen() or self.hitRock():
92                  self.initiateDeathAnimation()
93              #
94              # Increase score
95              self.score.value += interval/1000.0
96          elif self.animation.isComplete():
97              if self.keyboard.isClicked(pygame.K_KP_ENTER) or self.keyboard.isClicked(pygame.K_RETURN)
98                  self.restartGame()
99
100     def addSegment(self):
101         """Add a new body segment"""
102         segment = serge.actor.Actor('segment')
103         segment.setSpriteName('tail')
104         segment.setLayerName('middle')
```

```
105         segment.moveTo(self.x, self.y)
106         self.addChild(segment)
107         serge.sound.Sounds.play('new-body')
108 
109     def addRock(self):
110         """Add a rock to the screen"""
111         position = (random.randrange(0, 800), random.randrange(0, 600))
112         rock = serge.actor.Actor('rock')
113         rock.setSpriteName('rock')
114         rock.setLayerName('middle')
115         rock.moveTo(*position)
116         rock.setAngle(random.randrange(0, 360))
117         rock.linkEvent(serge.events.E_LEFT_CLICK, self.destroyRock, rock)
118         self.world.addActor(rock)
119 
120     def hitBody(self):
121         """Return True if the head has hit the body
122 
123         Look to see if we overlap with any body segment except the last
124         (we are allowed to overlap the last since we just put it down)
125 
126         """
127         for segment in self.getChildren()[:-1]:
128             if self.getDistanceFrom(segment) < 16:
129                 return True
130         return False
131 
132     def hitRock(self):
133         """Return True if we hit a rock"""
134         for rock in self.world.findActorsByTag('rock'):
135             if self.getDistanceFrom(rock) < 16:
136                 return True
137         else:
138             return False
139 
140     def offScreen(self):
141         """Return True if we are off the screen"""
142         return self.x < 0 or self.x > 800 or self.y < 0 or self.y > 600
143 
144     def initiateDeathAnimation(self):
145         """Begin showing the death of the snake"""
146         self.log.info('Snake died!')
147         self.animation = self.manager.assignBehaviour(self,
148             serge.blocks.behaviours.TimedCallback(1000/len(self.getChildren()), self.removeTail), 'de
149         self.is_dying = True
150         for segment in self.getChildren():
151             segment.setSpriteName('red-tail')
152         self.setSpriteName('red-head')
153         serge.sound.Sounds.play('snake-death')
154 
155     def removeTail(self, world, actor, interval):
156         """Remove part of the tail"""
157         self.log.debug('Removing part of the tail')
158         if self.getChildren():
159             self.removeChild(self.getChildren()[0])
160         else:
161             self.animation.markComplete()
162             self.restart_text.visible = True
```

```
            serge.sound.Sounds.getItem('snake-death').fadeout(500)

    def destroyRock(self, obj, rock):
        """Destroy a rock"""
        self.world.removeActor(rock)
        explosion = serge.blocks.actors.AnimateThenDieActor('explosion', 'explosion', 'explosion', '
        explosion.moveTo(rock.x, rock.y)
        self.world.addActor(explosion)

    def restartGame(self):
        """Restart the game"""
        self.is_dying = False
        self.restart_text.visible = False
        self.setSpriteName('head')
        self.current_direction = serge.blocks.directions.N
        self.score.value = 0
        self.moveTo(400, 300)
        self.world.clearActorsWithTags(['rock'])


# Create the engine
engine = serge.blocks.utils.getSimpleSetup(800, 600)
world = engine.getWorld('lab')

# Register sprites
serge.visual.Sprites.setPath('graphics')
serge.visual.Sprites.registerItem('head', 'head.png')
serge.visual.Sprites.registerItem('tail', 'tail.png')
serge.visual.Sprites.registerItem('red-head', 'red-head.png')
serge.visual.Sprites.registerItem('red-tail', 'red-tail.png')
serge.visual.Sprites.registerItem('rock', 'rock.png')
serge.visual.Sprites.registerItem('explosion', 'explosion.png', zoom=0.25,
        w=8, framerate=10, running=True, loop=False, one_direction=True)

# Register sounds
serge.sound.Sounds.setPath('sounds')
serge.sound.Sounds.registerItem('new-body', 'bloop.wav')
serge.sound.Sounds.registerItem('snake-death', 'death.wav')

# Register fonts
serge.visual.Fonts.setPath('fonts')
serge.visual.Fonts.registerItem('DEFAULT', 'MedievalSharp.ttf')
serge.visual.Fonts.registerItem('scores', 'PressStart2P.ttf')

# Create the snake
snake = Snake()
world.addActor(snake)
snake.moveTo(400, 300)

# Run the game
engine.run(60)
```

1.2.7 Conclusion

This concludes the snake tutorial. We have the basics of a playable game and covered the main fundamental concepts. We covered the following classes - take a look at their detailed documentation.

- The Engine
- Worlds
- Actors
- Sprites
- Sound
- Fonts
- Useful Blocks

Return to the *Tutorials* page for more advanced tutorial topics.

1.2.8 Resources

Here are the graphics, sounds and fonts needed for the game: zipfile

1.2.9 Credits

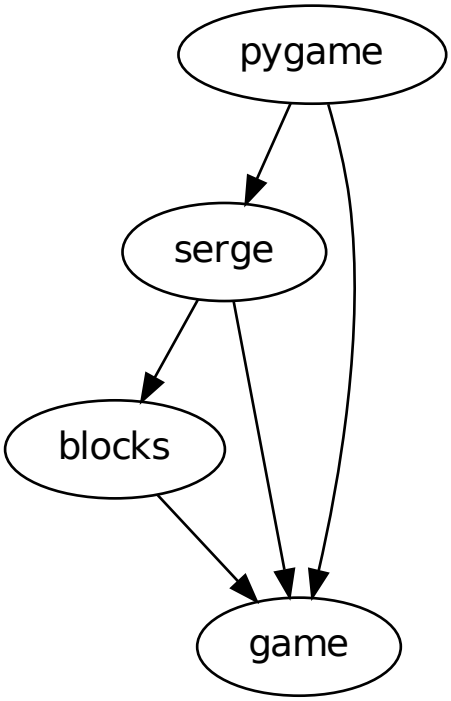
- http://www.freesound.org/people/Greencouch/sounds/124909
- http://www.freesound.org/people/suonho/sounds/3375
- http://openfontlibrary.org/en/font/press-start-2p
- http://openfontlibrary.org/en/font/medievalsharp

CHAPTER

TWO

USEFUL BUILDING BLOCKS

The serge engine is quite small and provides the core classes needed to construct a game. When building a game it is useful to also build upon some higher level pieces such as custom actor types, UI layouts and certain behaviours like responding to user input. This functionality is collected in the various *blocks* modules.



2.1 Building Blocks

2.1.1 `achievements` Module

Represents achievements

Achievements are badges that are assigned to the player as they play the game. An achievement is basically a condition that is met. When you meet the condition you get the badge.

class serge.blocks.achievements.**Achievement**(*name*, *description*, *badge*, *secret*, *test_type*, *condition=None*, *condition_string=None*)

Bases: serge.serialize.Serializable

Represents an achievement

index = 0

init()
Initialise the achievement from pickling

isMet()
Return True if the achievement was met

makeReport(***kw*)
Make a report on this achievement

my_properties = (‘’, ‘’, ‘’, 0, ‘’, <serge.serialize.Obj object at 0x4e61dd0>, ‘’, 0, 0.0)

class serge.blocks.achievements.**AchievementBanner**(*tag*, *name*, *background_layer*, *foreground_layer*, *behaviours*, *theme*)

Bases: serge.actor.MountableActor

A banner to show an achievment

addedToWorld(*world*)
Added the banner to the world

hideMe(*world*, *actor*, *interval*)
Hide ourself

meetAchievement(*achievement*, *arg*)
An achievement is met

class serge.blocks.achievements.**AchievementManager**

Bases: serge.serialize.Serializable, serge.common.Loggable, serge.common.EventAware

Manages all the achievements in the game

getAchievements()
Return the list of achievements

init()
Initialise

initialiseFromFile(*filename*)
Initialise from the file

log = <logging.Logger object at 0x4e61fd0>

makeReport(*test_type*, ***kw*)
Make a report on achievements

my_properties = (<serge.serialize.Obj object at 0x4e61e10>,)

registerAchievement(*achievement*)
Register an achievement

safeRegisterAchievement(*achievement*)
Register an achievement and do not worry if it is already registered

saveAchievements()
Save achievements to a file

class serge.blocks.achievements.**AchievementStatus**(*tag*, *name*, *background_layer*, *foreground_layer*, *achievement*, *G*)
Bases: serge.actor.MountableActor

A banner to show an achievment

addedToWorld(*world*)
Added the banner to the world

updateAchievement()
Update the achievement view

class serge.blocks.achievements.**AchievementsGrid**(*G*)
Bases: serge.blocks.actors.ScreenActor

A grid to show achievements

addedToWorld(*world*)
Added the grid to the world

updateAchievements(*obj*, *arg*)
Update all achievements

exception serge.blocks.achievements.**BadCondition**
Bases: exceptions.Exception

The condition was not valid

exception serge.blocks.achievements.**BadReport**
Bases: exceptions.Exception

An error occured while evaluating the report

exception serge.blocks.achievements.**BadTestType**
Bases: exceptions.Exception

The test type was not found

exception serge.blocks.achievements.**DuplicateAchievement**
Bases: exceptions.Exception

An achievement with this name already exists

serge.blocks.achievements.**addAchievementsBannerToWorld**(*world*, *front_layer*, *back_layer*, *theme*, *manager*)
Add a banner for achievements to the world

serge.blocks.achievements.**addAchievementsWorld**(*options*, *theme*)
Add a world for the achievements

serge.blocks.achievements.**getManager**()

serge.blocks.achievements.**initManager**(*name*)
Initialise and return the manager

2.1.2 actors Module

Blocks to help with actors

class serge.blocks.actors.**AnimateThenDieActor**(*tag*, *name*, *sprite_name*, *layer_name*, *parent=None*)

Bases: serge.actor.Actor

An actor that shows its animation and then is removed from the world

addedToWorld(*world*)
Added the actor to the world

updateActor(*interval*, *world*)
Update the actor

class serge.blocks.actors.**FPSDisplay**(*x*, *y*, *font_colour*, *font_size*, *font_name='DEFAULT'*)

Bases: serge.blocks.actors.NumericText

Displays the current FPS on the screen

addedToWorld(*world*)
Added to the world

updateActor(*interval*, *world*)
Update the actor

class serge.blocks.actors.**FocusManager**(*tag*, *name*)

Bases: serge.actor.CompositeActor

Manages focus between a number of entry widgets

actorEntry(*obj*, *actor*)
An entry was accepted

actorSelected(*obj*, *actor*)
An actor was selected

addChild(*actor*)
Add an actor to the manager

addedToWorld(*world*)
We were added to the world

updateActor(*interval*, *world*)
Update the manager

class serge.blocks.actors.**FormattedText**(*tag*, *name*, *format*, *colour*, *font_name='DEFAULT'*, *font_size=12*, *justify='center'*, ***kw*)

Bases: serge.actor.Actor

A text display that can be formatted

getValue(*name*)
Get the values

setValue(*name*, *value*)
Set the value

updateText()
Update our text

exception serge.blocks.actors.**InvalidMenu**

Bases: exceptions.Exception

The menu was not valid

exception serge.blocks.actors.**InvalidMenuItem**

Bases: exceptions.Exception

The menu item was not understood

class serge.blocks.actors.**MuteButton**(*sprite_name*, *layer_name*, *mute_sound=True*, *mute_music=True*, *alpha=1.0*)

Bases: serge.actor.Actor

A button to mute sound

toggleSound(*obj=None*, *arg=None*)
Clicked on the button

class serge.blocks.actors.**NumericText**(**args*, ***kw*)

Bases: serge.blocks.actors.FormattedText

A helper actor to display some text with a single number in there

updateText()
Update our text

value

class serge.blocks.actors.**RepeatedVisualActor**(*tag*, *name=None*, *repeat=5*, *spacing=10*, *orientation='horizontal'*)

Bases: serge.actor.Actor

An actor that shows multiple copies of a visual representation

This actor is useful for showing the number of lives or missiles etc in a game.

getRepeat()
Return the current repeat

increaseRepeat(*amount=1*)
Increase the repeat by a certain amount

reduceRepeat(*amount=1*)
Reduce the repeat by a certain amount

renderTo(*renderer*, *interval*)
Render ourself to the given renderer

resetRepeat()
Reset the repeat to the initial value

setRepeat(*value*)
Set the current repeat

class serge.blocks.actors.**ScreenActor**(**args*, ***kw*)

Bases: serge.actor.CompositeActor

An actor to represent the logic associated with a screen of the game

This actor is useful when encapsulating the logic associated with a specific screen in the game. The actor has useful properties and methods that make it easy to manage the logic.

addedToWorld(*world*)
The actor was added to the world

class serge.blocks.actors.**StringText**(*tag*, *name*, *text*, *format='%s'*, *colour=(255, 255, 255)*, *font_name='DEFAULT'*, *font_size=12*, *justify='center'*)

Bases: serge.blocks.actors.FormattedText

A helper actor to display some text with text in there

updateText()
Update our text

value

class serge.blocks.actors.**TextEntryWidget**(*tag*, *name*, *width*, *height*, *colour*, *font_size*, *font_name='DEFAULT'*, *justify='center'*, *background_visual=None*, *background_layer='background'*, *show_cursor=False*, *blink_time=0.5*, *has_focus=True*)

Bases: serge.actor.MountableActor

Implements a single line text entry widget

Support letters and numbers. Delete, backspace and left all delete the last character. Enter triggers an ACCEPT event.

addedToWorld(*world*)
Added to the world

getFocus()
Get the focus

getText()
Return the text value

hasFocus()
Return True if we have focus

loseFocus()
Lose the focus

setLayerName(*layer_name*)
Set the layer name

setText(*text*)
Set the text value

updateActor(*interval*, *world*)
Update the entry widget

class serge.blocks.actors.**ToggledMenu**(*tag*, *name*, *items*, *layout*, *default*, *on_colour*, *off_colour*, *width=100*, *height=100*, *callback=None*, *font_colour=(255, 255, 255, 255)*, *font_name='DEFAULT'*, *font_size=12*)

Bases: serge.actor.MountableActor

Implements a menu of options that can be toggled

getSelection()
Return the current selection

getSelectionIndex()
Return the current selection index

selectItem(*name*)
Select an item by name

selectItemIndex(*index*)
Select an item by its index

2.1.3 behaviours Module

Classes the implement behaviours

class serge.blocks.behaviours.**AvoidActor**(*actor*, *x_speed=1*, *y_speed=1*, *distance=10*)
Bases: serge.blocks.behaviours.Behaviour

Move away from an actor until you reach a certain distance

class serge.blocks.behaviours.**AvoidActorsWithTag**(*tag*, *x_speed=1*, *y_speed=1*, *distance=10*)
Bases: serge.blocks.behaviours.Behaviour

Move away from multiple actors until you read a certain distance

class serge.blocks.behaviours.**Behaviour**
Bases: serge.common.Loggable

Base class for all behaviours

exception serge.blocks.behaviours.**BehaviourAlreadyPaused**
Bases: exceptions.Exception

The behaviour was already paused

class serge.blocks.behaviours.**BehaviourManager**(**args*, ***kw*)
Bases: serge.actor.Actor

Manages the behaviour of multiple actors in a world

assignBehaviour(*actor*, *behaviour*, *name*)
Assign a behaviour to an actor

hasBehaviour(*record*)
Return True if we have a particular behaviour record

pauseBehaviours(*name*)
Pause all behaviours with the name

removeBehaviour(*record*)
Remove a particular behaviour

removeBehaviourByName(*actor*, *name*)
Remove the named behaviour for an actor based on its name

removeBehaviours(*behaviours*)
Remove a list of behaviours

removeBehavioursByName(*name*)
Remove the named behaviour for all actors based on a name

restartBehaviours(*name*)
Restart all behaviours with the name

updateActor(*world*, *interval*)
Perform all our behaviours

exception serge.blocks.behaviours.**BehaviourNotPaused**
Bases: exceptions.Exception

The behaviour was not paused

class serge.blocks.behaviours.**BehaviourRecord**(*actor*, *behaviour*, *name*)
Bases: object

Represents a record of a requested behaviour

involvesActor(*actor*)
Return True if the behaviour involves the actor

isComplete()
Return True if we are complete

isRunning()
Return True if we are running

markComplete()
Mark the behaviour as complete

The manage can now remove us and no longer try calling.

matches(*actor*, *name*)
Return True if this behaviour matches the actor and name

matchesName(*name*)
Return True if this behaviour matches the name

pause()
Pause the behaviour

performBehaviour(*interval*, *world*)
Perform the actual behaviour

restart()
Restart the behaviour

class serge.blocks.behaviours.**Blink**(*actor*, *time*)
Bases: serge.blocks.behaviours.Behaviour

Blink an actor on the screen

class serge.blocks.behaviours.**ConstantVelocity**(*vx*, *vy*)
Bases: serge.blocks.behaviours.Behaviour

Move an actor with a constant velocity

class serge.blocks.behaviours.**Delay**(*interval*)
Bases: serge.blocks.behaviours.TimedOneshotCallback

A delay - just waits and then completes

Usefor for sequences

exception serge.blocks.behaviours.**DuplicateBehaviour**
Bases: exceptions.Exception

The behaviour was already recorded

class serge.blocks.behaviours.**FlashFor**(*actor*, *time*)
Bases: serge.blocks.behaviours.Behaviour

Flash an actor on the screen

When the actor is flashing the property flashing is True.

class serge.blocks.behaviours.**KeyboardNSEW**(*speed*, *n=273*, *s=274*, *e=275*, *w=276*)
Bases: serge.blocks.behaviours.Behaviour

Move an actor in ordinal directions according to the keyboard

Set the n, s, e and w to the keys you want to move the actor. If you do not want any motion then set that direction to None. Set the speed to be the amount to move per keypress.

class serge.blocks.behaviours.**KeyboardNSEWToVectorCallback**(*method*, *event='keyclicked'*, *speed=1*, *n=273*, *s=274*, *e=275*, *w=276*)

Bases: serge.blocks.behaviours.Behaviour

Calls a method with direction vector from ordinal directions according to the keyboard

Set the n, s, e and w to the keys you want to move the actor. If you do not want any motion then set that direction to None. Set the speed to be the amount to move per keypress.

This is useful when you want to move an object but you need to do some preprocessing first. This behaviour will allow you to capture the keypresses.

class serge.blocks.behaviours.**KeyboardQuit**(*key=27*)

Bases: serge.blocks.behaviours.Behaviour

Quit the game based on a keypress

exception serge.blocks.behaviours.**MissingBehaviour**

Bases: exceptions.Exception

Could not locate the behaviour

class serge.blocks.behaviours.**MoveTowardsActor**(*actor*, *x_speed=1*, *y_speed=1*)

Bases: serge.blocks.behaviours.Behaviour

Move an actor towards another actor

class serge.blocks.behaviours.**MoveTowardsPoint**(*point*, *x_speed=1*, *y_speed=1*)

Bases: serge.blocks.behaviours.Behaviour

Move an actor towards a point

class serge.blocks.behaviours.**MoveWithMouse**(*actor*)

Bases: serge.blocks.behaviours.Behaviour

Move the actor with the mouse

class serge.blocks.behaviours.**OneShotSequence**(*sequence*)

Bases: serge.blocks.behaviours.Behaviour

A behaviour that calls a sequence of other behaviours

class serge.blocks.behaviours.**Optional**(*behaviour*, *arg*, *selector*)

Bases: serge.blocks.behaviours.TwoOptions

A behaviour that is turned on and off by an option

class serge.blocks.behaviours.**ParallaxMotion**(*parent*, *(sx*, *sy)*)

Bases: serge.blocks.behaviours.Behaviour

Move one object in relation to another

Parameters

- **parent** – the object to move relative to
- **sx** – fraction of x movement relative to parent (0.0 = no parallax, 1.0 = stationary)
- **sy** – fraction of y movement relative to parent

class serge.blocks.behaviours.**RemoveWhenOutOfRange**(*x_range*, *y_range*)

Bases: serge.blocks.behaviours.Behaviour

Remove an actor from the world when it is out of a certain range

class serge.blocks.behaviours.**SnapshotOnKey**(*key=115*, *size=(0, 0, 800, 600)*, *location=''*, *overwrite=True*)
Bases: serge.blocks.behaviours.Behaviour

Take a snapshot of the screen when the user presses a key

class serge.blocks.behaviours.**SpringTowardsPoint**(*point*, *spring_constant*, *damping*, *dead_zone=0.1*)
Bases: serge.blocks.behaviours.Behaviour

Move an actor towards a point as if on a spring

class serge.blocks.behaviours.**TimedCallback**(*interval*, *callback*)
Bases: serge.blocks.behaviours.Behaviour

A callback that gets called at a certain interval

class serge.blocks.behaviours.**TimedOneshotCallback**(*interval*, *callback*)
Bases: serge.blocks.behaviours.TimedCallback

A callback that gets called once and only once at an interval

class serge.blocks.behaviours.**TwoOptions**(*b1*, *b2*, *arg*, *selector*)
Bases: serge.blocks.behaviours.Behaviour

A behaviour that chooses between two optional behaviours

2.1.4 conversation Module

Represents a conversation

A conversation is a set of nodes with optional branches afterwards.

Each node has some text, either a single line or multiple lines.

exception serge.blocks.conversation.**BadOption**
Bases: exceptions.Exception

The option was not found

class serge.blocks.conversation.**ConversationManager**(*tree*, *callback=None*, *root=None*, *variables=None*)
Bases: serge.common.Loggable, serge.common.EventAware

Manages a conversation

chooseOption(*name*)
Choose an option named name

findNode(**names*)
Return the node with the given name

findNodeByID(*ID*)
Return the node with the given ID

classmethod fromXMLFile(*filename*)
Load from an XML filename

getChild()
Return the first child

getChildren()
Return the children

getLink()
Return a link or None if we don't have one

getNewManager(*parent*)
Return a new manager pointing to the parent

getNodeVariables(*node*)
Return the variables for this node

getParent()
Return the parent node

getText()
Return the text for this node

getVariable(*name*)
Return the value of a variable

moveNext()
Move on to the next node

parseRichText(*lines*, *data*)
Return the variables and text from a rich text list

processNodeVariables(*node*)
Process variables in the current node

restartConversation()
Try to restart the conversation

setCallback(*callback*)
Set the callback

exception serge.blocks.conversation.**InvalidFile**
Bases: exceptions.Exception

The XML file was not valid

exception serge.blocks.conversation.**NodeNotFound**
Bases: exceptions.Exception

The node was not found in the tree

2.1.5 directions Module

Utilities to do with cardinal directions

serge.blocks.directions.**getAngleFromCardinal**(*direction*)
Return the angle for a cardinal direction

serge.blocks.directions.**getCardinalFromAngle**(*angle*)
Return the cardinal for an angle

serge.blocks.directions.**getCardinalFromVector**(*vector*)
Return the cardinal name from the vector

serge.blocks.directions.**getCardinals**()
Return the cardinal directions by name

serge.blocks.directions.**getOppositeCardinal**(*cardinal*)
Return the opposite cardinal direction

serge.blocks.directions.**getOppositeVector**(*vector*)
Return the opposite vector

serge.blocks.directions.**getVectorFromCardinal**(*direction*)
Return the vector for a cardinal direction

2.1.6 dragndrop Module

Implements drag and drop behaviour

exception serge.blocks.dragndrop.**AlreadyATarget**
Bases: exceptions.Exception

The actor is a target already

class serge.blocks.dragndrop.**DragController**(*tag='controller'*, *name='controller'*, *start=None*, *stop=None*, *hit=None*, *miss=None*)
Bases: serge.blocks.actors.ScreenActor

Controls objects which are draggable

addActor(*actor*, *start=None*, *stop=None*)
Add an actor to be controlled and callback to be called when dragging start and stops

addDropTarget(*actor*, *fn=None*)
Add a target to drop to

checkForDrops(*actor*)
Check to see if we dropped our actor onto a target or not - return False if the drop is not allowed

If we dropped on a target then we can call the callback. If we didn't drop on a target then we call the miss callback.

The callback can raise DropNotAllowed to cause the drop not to occur

clickedActor(*obj*, *(actor*, *fn)*)
The mouse was released over an actor

getDraggedActor()
Return the actor being dragged

isDragging()
Return True if we are dragging an object

mouseDown(*obj*, *(actor*, *fn)*)
The mouse was down over an actor

removeActor(*actor*)
Remove an actor from being controlledd

removeDropTarget(*actor*)
Remove an actor as a drop target

setCallbacks(*start*, *stop*)
Set the callbacks to use when starting and stopping a drag

setDropCallbacks(*hit*, *miss*)
Set the callback to use when dropping on a target

updateActor(*interval*, *world*)
Update the controller

exception serge.blocks.dragndrop.**DropNotAllowed**
Bases: exceptions.Exception

Cannot drop here

exception serge.blocks.dragndrop.**DuplicateActor**
Bases: exceptions.Exception

The actor is already controlled

exception serge.blocks.dragndrop.**NotATarget**
Bases: exceptions.Exception

The actor is not a target

exception serge.blocks.dragndrop.**NotDragging**
Bases: exceptions.Exception

No actor is being dragged

2.1.7 effects Module

Some effects which can alter properties of actors or visuals

class serge.blocks.effects.**AttributeFade**(*obj*, *attribute_name*, **args*, ***kw*)
Bases: serge.blocks.effects.MethodCallFade

Linearly move an attribute

The attribute changes between a start and an end with a decay. The decay is the length of time taken to get from the start to the end.

If persistent is set to true then the effect remains in the world to be re-used. If false then it will be removed when completed.

class serge.blocks.effects.**ColourPhaser**(*red*, *green*, *blue*, **args*, ***kw*)
Bases: serge.blocks.effects.Effect

An effect that causes colours on the whole screen to fade in and out

addedToWorld(*world*)
We were added to a wolrd

postRender(*obj*, *arg*)
Update this effect

worldChange(*obj*, *active*)
The world changed its state

Since we are linked to a renderer event we will be called from all worlds but we only want to be active when our world is the ative one. We use the world activation events to toggle our state.

class serge.blocks.effects.**Effect**(*done=None*, *persistent=False*)
Bases: serge.actor.Actor

A generic effect

finish()
End the effect

pause()
Pause the effect

restart()
Restart the effect

unpause()
Unpause the effect

exception serge.blocks.effects.**InvalidMotion**
Bases: exceptions.Exception

The motion type was not recognized

class serge.blocks.effects.**MethodCallFade**(*method*, *start*, *end*, *decay*, *persistent=False*, *done=None*, *motion='linear'*)
Bases: serge.blocks.effects.Effect

Repeated call a method linearly changing the parameter over time

The attribute changes between a start and an end with a decay. The decay is the length of time taken to get from the start to the end.

If persistent is set to true then the effect remains in the world to be re-used. If false then it will be removed when completed.

A method can be provided through the done parameter which will be called when the effect has completed.

The way the variable is moved is dependent on the motion type. This can be 'linear' or 'accelerated'.

updateActor(*interval*, *world*)
Update this effect

class serge.blocks.effects.**PanActor**(*actor*, *speed*, *done=None*, *persistent=False*, *linear=True*)
Bases: serge.blocks.effects.Effect

Pan an actor across the screen

restart()
Restart the panning

updateActor(*interval*, *world*)
Update the panning

class serge.blocks.effects.**Pause**(*time*, *done*, *persistent=False*)
Bases: serge.blocks.effects.Effect

A simple pause

Used in conjunction with other effects. Calls the done method when the pause has completed.

updateActor(*interval*, *world*)
Update this effect

2.1.8 fractals Module

Some fractal utilities

serge.blocks.fractals.**fractalLine**(*start*, *end*, *number_steps*, *distance_per_step*, *decay*)
Return a fractal line, broken into # steps with each step being a random distance

serge.blocks.fractals.**fractalShape**(*points*, *number_steps*, *distance_per_step*, *decay*)
Return a shape where the straight lines are converted to fractal lines

2.1.9 layout Module

Blocks to help with laying out things on the screen

exception serge.blocks.layout.**AlreadyInCell**
Bases: exceptions.Exception

The actor was already in this cell

class serge.blocks.layout.**Bar**(*tag*, *name=''*, *width=None*, *height=None*, *background_colour=None*, *background_layer=None*, *background_sprite=None*)
Bases: serge.blocks.layout.Container

A bar of actors - useful for user interfaces

addActor(*actor*, *layer_name=None*)
Add an actor to the bar

addBlanks(*number*)
Add blank entries into the bar

class serge.blocks.layout.**BaseGrid**(*tag*, *name=''*, *size=(1, 1)*, *width=None*, *height=None*, *background_colour=None*, *background_layer=None*)
Bases: serge.blocks.layout.Container

A grid of actors

clearGrid()
Clear the entire grid

getCoords(*(x, y)*)
Return the coordinates of a location

getLocation(*(x, y)*)
Return the location in the grid based on coordinates

removeActor(*(x, y)*)
Remove the actor at a certain location

removeChildren()
Remove all the children

setGrid(*(w, h)*)
Set the size of the grid

This also removes all the current actors from the world. Note that this can be tricky if you want to re-add some of the actors since the actors are not actually removed until the next world update and so you cannot re-add them before this or you will get a duplicate actor error from the world.

exception serge.blocks.layout.**CellEmpty**
Bases: exceptions.Exception

The cell being accessed was empty

exception serge.blocks.layout.**CellOccupied**
Bases: exceptions.Exception

Tried to put an actor in an occupied cell

class serge.blocks.layout.**Container**(*tag*, *name=''*, *width=None*, *height=None*, *background_colour=None*, *background_layer=None*, *background_sprite=None*)
Bases: serge.actor.MountableActor

A layout container that contains actors

moveTo(*x*, *y*, *no_sync=False*, *override_lock=True*)
Move this actor

reflowChildren()
Relocate all children

setBackgroundColour(*colour*)
Sets the background colour

setBackgroundSprite(*name*)
Sets the background sprite

setLayerName(*name*)
Set the layer name

class serge.blocks.layout.**Grid**(*tag*, *name=''*, *size=(1, 1)*, *width=None*, *height=None*, *background_colour=None*, *background_layer=None*)
Bases: serge.blocks.layout.BaseGrid

A grid where a cell can only contain a single actor

addActor(*(x, y)*, *actor*, *layer_name=None*)
Add an actor to the grid

autoAddActor(*actor*)
Automatically add an actor to the next cell in the grid

This fills horizontally and then vertically

findActorLocation(*actor*)
Find the location of an actor

getActorAt(*(x, y)*)
Return the actor at a certain location

moveActor(*(x, y)*, *actor*)
Move an actor from wherever it is to the new location

class serge.blocks.layout.**HorizontalBar**(*tag*, *name=''*, *width=None*, *height=None*, *background_colour=None*, *background_layer=None*, *background_sprite=None*)
Bases: serge.blocks.layout.Bar

A horizontal bar of actors

getCoords(*i*)
Return the coordinates of our ith location

class serge.blocks.layout.**MultiGrid**(*tag*, *name=''*, *size=(1, 1)*, *width=None*, *height=None*, *background_colour=None*, *background_layer=None*)
Bases: serge.blocks.layout.BaseGrid

A grid where each cell can contain multiple actors

addActor(*(x, y)*, *actor*, *layer_name=None*)
Add an actor to the grid

findActorLocation(*actor*)
Find the location of an actor

getActorsAt(*(x, y)*)
Return the actors at a certain location

moveActor(*(x, y)*, *actor*)
Move an actor from wherever it is to the new location

removeActor(*(x, y), actor*)
Remove the actor at a certain location

removeActors(*(x, y)*)
Remove all the actor from a certain location

exception serge.blocks.layout.**OutOfRange**
Bases: exceptions.Exception

Tried to find something outside the range of the container

exception serge.blocks.layout.**UnknownActor**
Bases: exceptions.Exception

The actor was not found

class serge.blocks.layout.**VerticalBar**(*tag, name=''*, *width=None*, *height=None*, *background_colour=None*, *background_layer=None*, *background_sprite=None*)
Bases: serge.blocks.layout.Bar

A vertical bar of actors

2.1.10 scores Module

Handling high score type tables

exception serge.blocks.scores.**BadCategory**
Bases: exceptions.Exception

The category was not found

exception serge.blocks.scores.**BadData**
Bases: exceptions.Exception

The data provided for a category was not valid

class serge.blocks.scores.**Category**(*name*, *number=None*, *sort_columns=None*, *directions=('ascending',)*)
Bases: list

A category for an individual score table

addScore(*name*, **args*)
Add a new score

resetCategory()
Reset this category, deleting all the data but maintaining the configuration

exception serge.blocks.scores.**DuplicateCategory**
Bases: exceptions.Exception

The category was already added

class serge.blocks.scores.**HighScoreTable**
Bases: serge.serialize.Serializable

A high score table

The table can contain scores in a number of categories. Each category is a table with multiple columns. The table can be sorted by any one column and can have a limited set of values

addCategory(*name*, *number=None*, *sort_columns=None*, *directions=('ascending',)*)
Add a new category

addScore(*category_name*, *name*, **args*)
Add a score to a category

getCategory(*category_name*)
Return a category

my_properties = ({},)

resetCategory(*category_name*)
Reset the category name

resetTable()
Clear the entire table

exception serge.blocks.scores.**InvalidSort**
Bases: exceptions.Exception

The sort direction was invalid

exception serge.blocks.scores.**InvalidSortColumn**
Bases: exceptions.Exception

The column specified for sorting was not valid

2.1.11 singletons Module

Implement a store for singletons

class serge.blocks.singletons.**SingletonStore**
Bases: serge.registry.GeneralStore

A store for global objects

2.1.12 themes Module

Classes to implement themes

Themes are sets of settings that may affect anything. The idea is that you may have a number of settings to do with visuals on a world and you want to control those centrally, potentially also allowing things to switch during a game.

The themes are managed by a manager.

exception serge.blocks.themes.**BadInheritance**
Bases: exceptions.Exception

A theme subclass was not found

exception serge.blocks.themes.**BadThemeDefinition**
Bases: exceptions.Exception

The theme was not of the right format

exception serge.blocks.themes.**BadThemeFile**
Bases: exceptions.Exception

The specified theme file was not found

exception serge.blocks.themes.**InvalidFormat**
Bases: exceptions.Exception

The format for the data was invalid

class serge.blocks.themes.**Manager**
Bases: object

Manages a theme

getProperty(*name*, *from_theme=None*)
Return the named property

getPropertyWithDefault(*name*, *default*, *from_theme=None*)
Return a property and if it is missing then return the default value

Use this method sparingly. It puts default values in source code rather than in the theme files.

getTheme(*name*)
Return a theme object with a default of the given name

hasTheme(*name*)
Return True if we have this theme

load(*themes*)
Load definitions from a dictionary

loadFrom(*text*)
Load a theme from some text

The theme is a dictionary where each entry is either a theme or the definition of the schema or the special entry __default__, which gives the name of the default theme.

If there is an entry then it is a tuple with the name of the base theme class followed by a dictionary of entries which overide the base class.

Classes are really just the name of another theme.

loadFromFile(*filename*)
Load a theme definition from a file

selectTheme(*name*)
Select the named theme

setProperty(*name*, *value*, *from_theme=None*)
Set a property in a theme

updateFromString(*string*)
Update the theme from a string of data

Data should be provided as comma separated values like name="bob",value=123,etc

exception serge.blocks.themes.**MissingDefault**
Bases: exceptions.Exception

There was no default theme

exception serge.blocks.themes.**MissingSchema**
Bases: exceptions.Exception

There was no schema in the theme definition

exception serge.blocks.themes.**PropertyNotFound**
Bases: exceptions.Exception

Could not find a property

exception serge.blocks.themes.**ThemeNotFound**
Bases: exceptions.Exception

The named scheme was not found

2.1.13 `tiled` Module

Implements an interface to Tiled files

exception serge.blocks.tiled.**BadLayer**
Bases: exceptions.Exception

The layer specification was invalid

exception serge.blocks.tiled.**BadTiledFile**
Bases: exceptions.Exception

The tiled file could not be found

class serge.blocks.tiled.**Layer**(*tiled*, *name*, *layer_type*, *width=None*, *height=None*, *tiles=None*, *properties=None*)
Bases: serge.common.Loggable

A layer in a tilemap

addObject(*obj*)
Add an object

getLocationsWithTile()
Return all tile locations with a tile

getLocationsWithoutTile()
Return all tile locations without a tile

getObject(*name*)
Return the named object

getObjects()
Return all the objects

getSize()
Return the size of the layer

getSpriteFor(*(x, y)*)
Return the sprite for a certain location

iterCellLocations()
Return an interation of the cell locations

exception serge.blocks.tiled.**NotFound**
Bases: exceptions.Exception

The object was not found

class serge.blocks.tiled.**TileMap**
Bases: serge.common.Loggable

A representation of a 2d map of tiles

addLayer(*layer*)
Add a layer

classmethod addLayerTypes(*layer_types*)
Add more layer types

getLayer(*name*)
return the tile with a certain name

getLayerByType(*type_name*)
Return the layer with a given type

getLayers()
Return the layers of tiles

getLayersByType(*type_name*)
Return the layer with a given type

getLayersForTile((*x*, *y*), *excluding=None*)
Return a list of the layers that the tile at x, y is set on

getPropertiesFrom(*nodes*)
Return a property disction from the node

getPropertyBagArray(*sprite_layers*, *boolean_layers*, *property_layers*, *prototype=None*, *optional_layers=None*)
Return an array of property bags for the tile array

You pass a series of lists of layer types, which are treated like: sprite_layers = tile based layers to treat as identifying sprites boolean_layers = tile layers where if a tile is set (to anything) then a boolean flag is True property_layers = tile layers where if a tile is set then the item recieves all the properties of the layer

getSize()
Return the size of the map using the first layer as a guide

getSpriteName(*idx*)
Return the sprite name for an index

getTypeFrom(*name*, *properties*)
Return the layer type, checking validity which we do it

classmethod resetLayerTypes()
Reset the layer types to default

class serge.blocks.tiled.**TileObject**(*name*, *object_type*, *x*, *y*, *width*, *height*, *properties*, *sprite_name=None*)
Bases: serge.common.Loggable

A tile

class serge.blocks.tiled.**Tiled**(*filename*)
Bases: serge.blocks.tiled.TileMap

An interface to tiled files

getObjectLayers()
Return the object layers

layer_types = ['visual', 'adhoc-visual', 'movement', 'visibility', 'object', 'resistance']

2.1.14 utils Module

Some utilities that speed up common operations

class serge.blocks.utils.**MovieRecorder**(*path*, *make_movie=False*, *rate=1*, *in_memory=False*)
Bases: object

Will record a movie of the game

clearFrames()
Clear all current frames

makeFrame(*obj*, *arg*)
Make a frame

makeMovie(*obj*, *arg*)
Convert the frames to movie

class serge.blocks.utils.**RecordDesktop**(*filename*)
Bases: serge.common.Loggable

Use record my desktop to record the action

stop(*obj*, *arg*)
Stop the recording

serge.blocks.utils.**addActorToWorld**(*world*, *actor*, *sprite_name=None*, *layer_name=None*, *center_position=None*, *physics=None*, *origin=None*)
Create a new actor in the world

If the center position is not specified then it is placed at the center of the screen.

serge.blocks.utils.**addMuteButtonToWorlds**(*button*, *center_position*, *world_names=None*)
Add a particular mute button to various worlds

If worlds is not specified then add to all the worlds currently in the engine.

serge.blocks.utils.**addSpriteActorToWorld**(*world*, *tag*, *name*, *sprite_name*, *layer_name*, *center_position=None*, *physics=None*)
Create a new actor in the world and set the visual to be the named sprite

If the center position is not specified then it is placed at the center of the screen.

serge.blocks.utils.**addTextItemsToWorld**(*world*, *items*, *theme*, *layer_name*)
Add multiple text items to the world

serge.blocks.utils.**addTextToWorld**(*world*, *text*, *name*, *theme*, *layer_name*)
Add some text to the world

serge.blocks.utils.**addVisualActorToWorld**(*world*, *tag*, *name*, *visual*, *layer_name*, *center_position=None*, *physics=None*)
Create a new actor in the world and set the visual

If the center position is not specified then it is placed at the center of the screen.

serge.blocks.utils.**backToPreviousWorld**(*sound=None*)
Return an event callback to switch back to the previous world

serge.blocks.utils.**checkNetworkXVersion**(*need_version*)
Check a suitable version of NetworkX is installed

serge.blocks.utils.**checkPythonVersion**()
Check a suitable Python version is installed

serge.blocks.utils.**createLayers**(*engine*, *layers*, *cls*)
Create a number of layers in the engine using the given class of layer

serge.blocks.utils.**createLayersForEngine**(*engine*, *layers*)
Add a number of layers to the engine

The layers parameter is a list of layer names. The layers are added to the renderer of the engine as successive layers in order.

serge.blocks.utils.**createVirtualLayersForEngine**(*engine*, *layers*)
Add a number of virtual layers to the engine

The layers parameter is a list of layer names. The layers are added to the renderer of the engine as successive layers in order.

The layers are created as virtual, meaning that this will render quicker than the real layers version, although compositing will not be possible.

serge.blocks.utils.**createWorldsForEngine**(*engine*, *worlds*)
Add a numer of worlds to the engine

The words parameter is a list of names of the worlds to create. Each world is created with a single active zone which is quite large.

serge.blocks.utils.**debugMethod**(*obj*, *method_name*, *logger=None*, *fmt=''*)
Create a debug logged method

serge.blocks.utils.**getGamePath**(**parts*)
Return a path based on the main game folder

serge.blocks.utils.**getSimpleSetup**(*width*, *height*)
Return an engine with a single world, zone and a few layers

serge.blocks.utils.**worldCallback**(*name*, *sound=None*)
Return an event callback to switch to a certain world

2.1.15 visualblocks Module

Useful blocks for visual rendering

class serge.blocks.visualblocks.**Circle**(*radius*, *colour*, *stroke_width=0*, *stroke_colour=None*)
Bases: serge.visual.SurfaceDrawing

A circle

colour

radius

setAngle(*angle*)
Set the angle

Pass through as this is a circle!

class serge.blocks.visualblocks.**CircleText**(*text*, *text_colour*, *radius*, *circle_colour*, *font_size=12*, *font_name='DEFAULT'*, *stroke_width=0*, *stroke_colour=None*, *justify='center'*)
Bases: serge.visual.Drawing

A circle with some text on it

getSize()
Return the size of the drawing

renderTo(*milliseconds*, *surface*, *(x*, *y)*)
Render to a surface

exception serge.blocks.visualblocks.**InvalidParameters**
Bases: exceptions.Exception

The parameters for the shape were not valid

exception serge.blocks.visualblocks.**InvalidSprite**
Bases: exceptions.Exception

The selected sprite was not valid

exception serge.blocks.visualblocks.**OutOfRange**
Bases: exceptions.Exception

The value was outside the valid range

exception serge.blocks.visualblocks.**OverlappingRanges**
Bases: exceptions.Exception

The ranges for the progress bar were overlapping

class serge.blocks.visualblocks.**ProgressBar**(*size*, *value_ranges*, *border_width=0*, *border_colour=(255, 255, 255, 255)*)
Bases: serge.visual.SurfaceDrawing

A progress bar

The progress bar shows a rectangle on the screen which you can use to show progress or represent the number of cetain items. The bar can be a single colour or can change colour within certain ranges.

value

exception serge.blocks.visualblocks.**RangesNotContiguous**
Bases: exceptions.Exception

The ranges for the progress bar had gaps in them

class serge.blocks.visualblocks.**Rectangle**(*(w, h)*, *colour*, *stroke_width=0*, *stroke_colour=None*)
Bases: serge.visual.SurfaceDrawing

A rectangle

colour

class serge.blocks.visualblocks.**RectangleText**(*text*, *text_colour*, *rect_dimensions*, *rect_colour*, *font_size=12*, *font_name='DEFAULT'*, *stroke_width=0*, *stroke_colour=None*, *justify='center'*)
Bases: serge.visual.Drawing

A rectangle with some text on it

renderTo(*milliseconds*, *surface*, *(x, y)*)
Render to a surface

class serge.blocks.visualblocks.**SpriteText**(*text*, *text_colour*, *sprite_name*, *font_size=12*, *font_name='DEFAULT'*, *stroke_width=0*, *stroke_colour=None*, *justify='center'*)
Bases: serge.visual.Sprite

A sprite with some text on it

renderTo(*milliseconds*, *surface*, *(x, y)*)
Render to a surface

setText(*text*)
Set the text

text

class serge.blocks.visualblocks.**TextToggle**(**args*, ***kw*)
Bases: serge.blocks.visualblocks.SpriteText

A sprite text item that has multiple cells and can be used as a toggle

You can set the cells directly of use On=0 and Off=1.

isOff()
Return if we are on

isOn()
Return if we are on

setOff()
Set to off

setOn()
Set to on

toggle()
Toggle the state

class serge.blocks.visualblocks.**Toggle**(*sprite_name*)
Bases: serge.blocks.visualblocks.TextToggle

Like a text toggle but with no text

2.1.16 visualeffects Module

Visual effects

class serge.blocks.visualeffects.**FadingLayer**(*name*, *order*)
Bases: serge.render.Layer

A layer that you can fade in and out

postRender()
After rendering the surface

class serge.blocks.visualeffects.**FadingScreen**
Bases: object

Fade in and out everything

deleteFade()
Remove the fade

postRender(*obj*, *arg*)
After rendering the surface

class serge.blocks.visualeffects.**Shadow**(*source*, *colour*)
Bases: serge.visual.SurfaceDrawing

Creates a shadow from an image

createShadow()
Create the shadow now

Most of the logic here from http://pygame.org/wiki/ShadowEffects

class serge.blocks.visualeffects.**ShadowLayer**(*name*, *order*, *colour*, *offset*)
Bases: serge.render.Layer

A layer that renders with a shadow beneath it

initSurface(*renderer*)
Initialise the surface

render(*surface*)
Render to a surface

When rendering to the surface we first create our shadow then render this to the surface followed by our normal rendering.

serge.blocks.visualeffects.**darkenSurf**(*img*, *amount*)
Darken a surface

serge.blocks.visualeffects.**darkenSurf2**(*img*, *amount*)
Darken the given surface by the given amount

serge.blocks.visualeffects.**fadeSurface**(*surface*, *v*)
Fade the given suface by an amount 0 to 255 - 0 is completely faded

serge.blocks.visualeffects.**gaussianBlur**(*surface*, *sigma*)
This function takes a pygame surface, converts it to a numpy array carries out gaussian blur, converts back then returns the pygame surface.

2.1.17 worker Module

Some helper classes to implement various parallel processing workers

serge.blocks.worker.**SkippableQueue**()
Return A queue where only one item is retained

serge.blocks.worker.**getSurfaceProcessingPipeline**(*target*, *start=True*)
Return a pair of queues to implement a surface processing pipeline

An input and output queue are returned. The queues are passed a tuple of items and the first one is a surface which is marshalled to the target function.

The function must also return a tuple, the first of which is assumed to be a surface which will be marshalled.

serge.blocks.worker.**marshallSurface**(*surface*)
Return a surface that can be passed from one process to another

serge.blocks.worker.**pipelineProcessor**(*qin*, *qout*, *target*)
Implements the surface processing pipeline

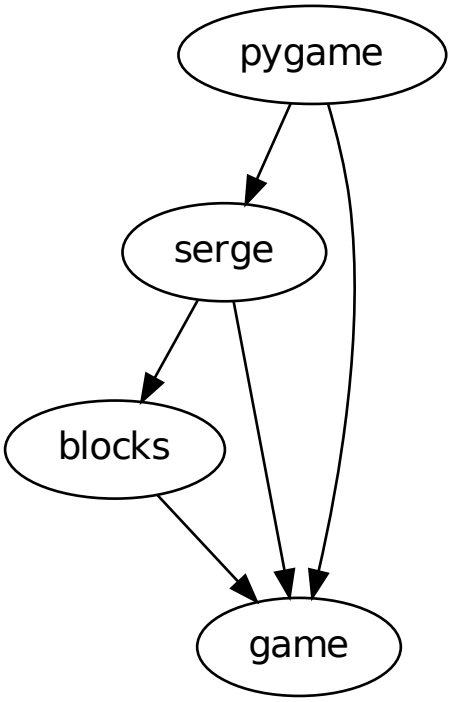
serge.blocks.worker.**unmarshallSurface**(*width*, *height*, *fmt*, *string*)
Return a surface returned from another process

CHAPTER

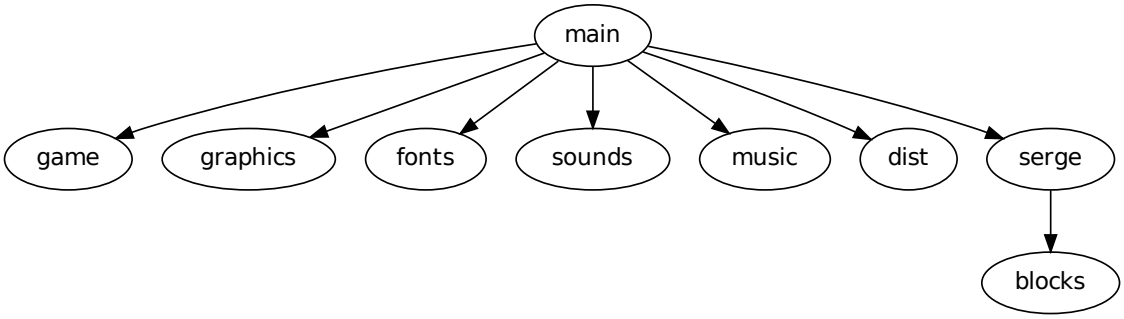
THREE

ENGINE OVERVIEW

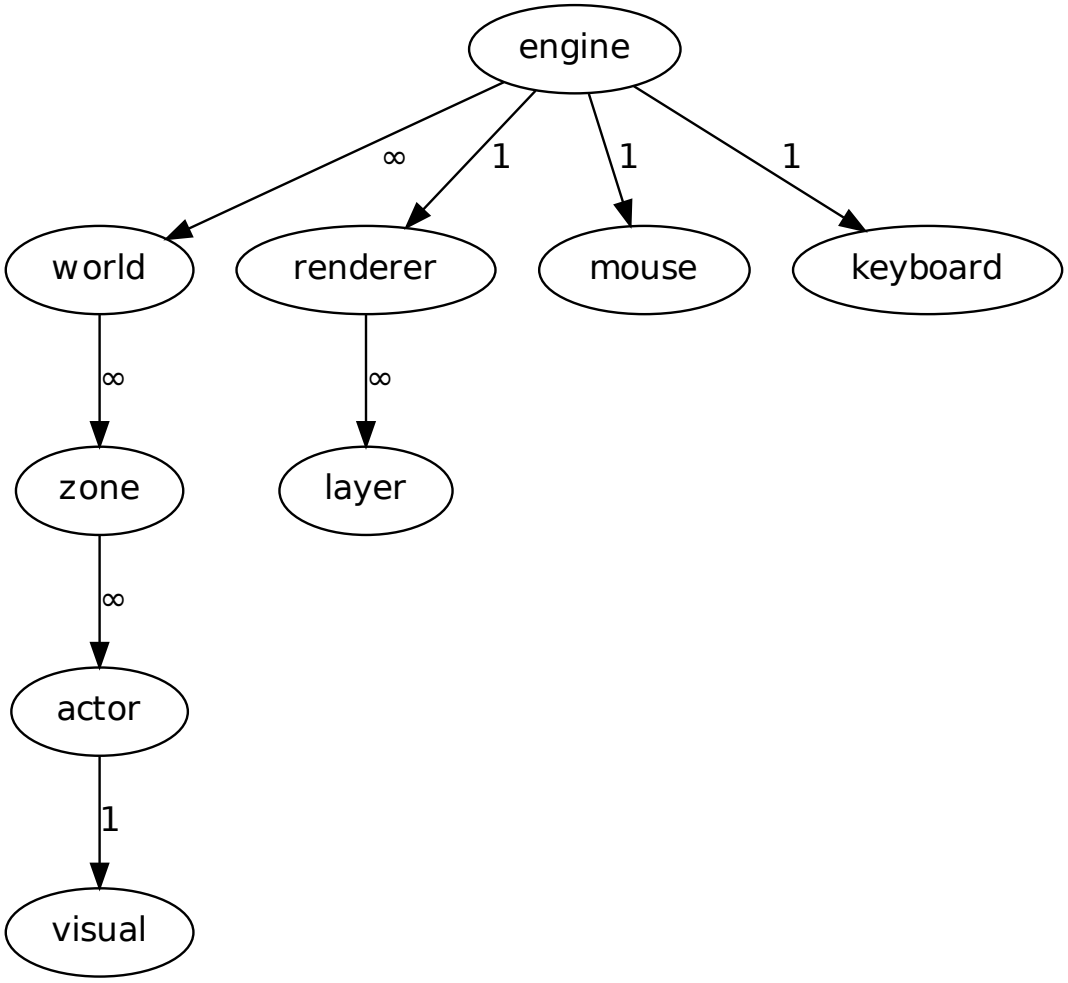
Serge is a game engine writen in Python on top of pygame.



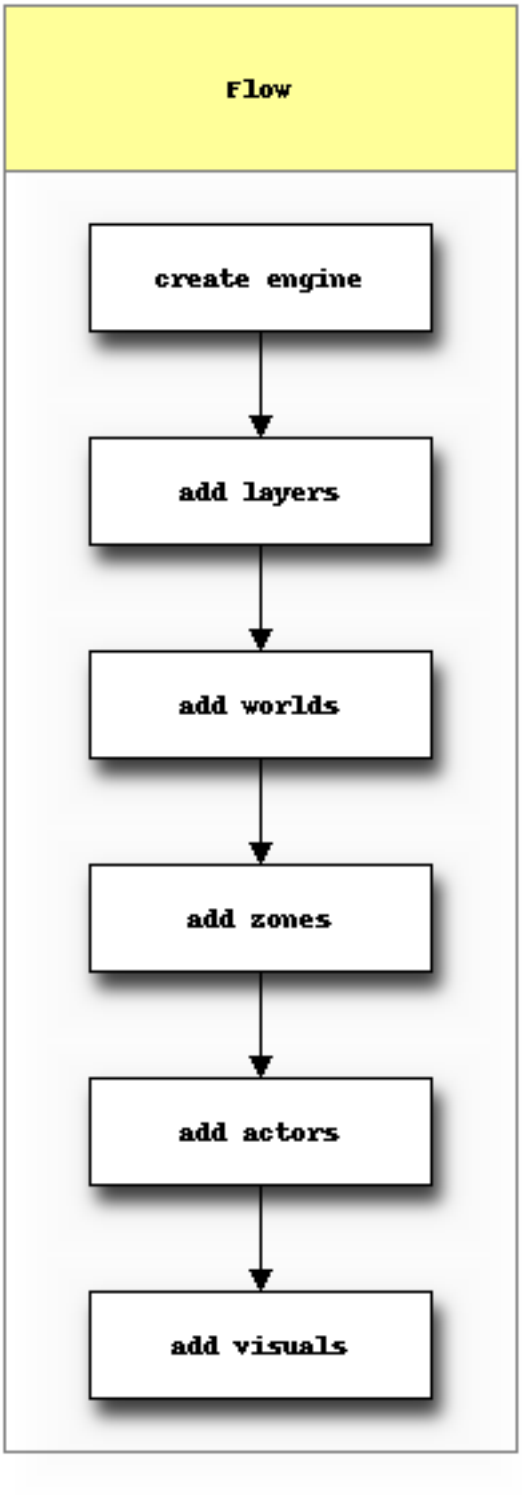
Typical game folder structure for a game.



Engine Structure.



Typical game flow.



Class Structures:

CHAPTER

FOUR

ENGINE

Contents

4.1 Engine

class serge.engine.**Engine**(*width=640*, *height=480*, *title='Serge'*, *backcolour=(0, 0, 0)*, *icon=None*, *fullscreen=False*)

Bases: serge.common.Loggable, serge.serialize.Serializable, serge.common.EventAware

The main Serge engine

The engine manages a set of worlds and allows a single *Worlds*, the current world, to be automatically updated on a certain time frequency.

addWorld(*world*)
Add a world to the engine

Parameters **world** – the world instance to add

attachBuilder(*builder*)
Attach a builder

clearWorlds()
Clear all the worlds

detachBuilder()
Detach the builder

getCurrentWorld()
Return the currently selected world

getKeyboard()
Return the keyboard

getMouse()
Return the mouse

getRenderer()
Return the renderer

getSprites()
Return the sprite registry

getStats()
Return the stats for the engine

getWorld(*name*)
Return the named world

Parameters **name** – the name of the world to return

getWorlds()
Return all the worlds

goBackToPreviousWorld(*obj=None*, *arg=None*)
Return to the world we were in before this one

The arguments are never used and are just here to allow you to use this method as an event callback.

init()
Initialise ourself

processEvents()
Process all the events for the current world

removeWorld(*world*)
Remove a world from the engine

Parameters **world** – the world instance to remove

removeWorldNamed(*name*)
Remove a world with a given name

Parameters **name** – the name of the world to remove

run(*fps*, *endat=None*)
Run the updates at the specified frames per second until the optional endtime

Parameters

- **fps** – the target frames per second (integer)
- **endat** – a time to stop the engine at (long), eg time.time()+60 to run for a minute

runAsync(*fps*, *endat=None*)
Run the engine asynchronously

Parameters

- **fps** – the target frames per second (integer)
- **endat** – a time to stop the engine at (long), eg time.time()+60 to run for a minute

save(*filename*)
Store the engine state in a file suitable for loading again in the furture

Parameters **filename** – the name of the file to save into

setCurrentWorld(*world*)
Set the current world

Parameters **world** – the world to set as the current world

setCurrentWorldByName(*name*)
Set the current world to the one with the given name

Parameters **name** – the name of the world to set as the current world

stop()
Stop the engine running

updateWorld(*interval*)
Update the current world

4.2 EngineStats

class serge.engine.**EngineStats**
Statistic for the engine

afterRender()
Record that we are after a rendering cycle

beforeRender()
Record we are before a rendering cycle

recordFrame()
Record a frame

CHAPTER

FIVE

WORLDS

5.1 Zones

class serge.zone.**Zone**

Bases: serge.geometry.Rectangle, serge.common.Loggable

A zone

A zone is part of a world. It is a container for objects and it controls whether objects will take part in world updates.

addActor(*actor*)
Add an actor to the zone

clearActors()
Remove all actors

findActorByName(*name*)
Return the actor with the given name

findActorsByTag(*tag*)
Return all the actors with a certain tag

findFirstActorByTag(*tag*)
Return the first actor found with the given tag or raise an error

getActors()
Return all the actors

hasActor(*actor*)
Return True if the actor is in this zone

init()
Initialise from serialized state

removeActor(*actor*)
Remove an actor from the zone

setGlobalForce(*force*)
Set the global force for physics

setPhysicsStepsize(*interval*)
Set the maximum step size for physics calculations

sleepActor(*actor*)
Tell the actor to go to sleep from a physics perspective

The actor will still be visible and will still be updated but it will not update its physics. Useful for optimising when an actor does not need to interact with the physics simulation for a while.

updatePhysics (*interval*)
Perform a step of the physics engine

You do not normally need to call this method as it is called by the updateZone method. You may call this to advance the physics simulation along without affecting other game elements.

updateZone (*interval*, *world*)
Update the objects in the zone

wakeActor (*actor*)
Tell the actor to go to wake up from a physics perspective

An actor that was put to sleep (via sleepActor) will be woken up and take part in the physics simulation again.

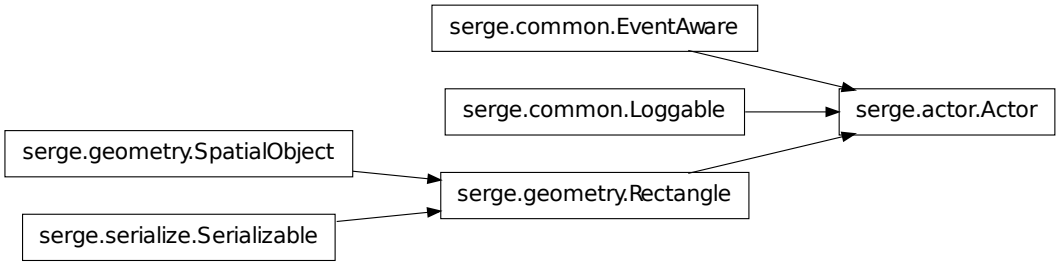
wouldContain (*actor*)
Return True if this zone would contain the actor as it is right now

The base Zone implementation uses spatial overlapping as the criteria but you can create custom zones that use other criteria to decide which actors should be in the zone.

5.2 Actors

Contents

5.2.1 Actor



class serge.actor.**Actor** (*tag*, *name=''*)
Bases: serge.common.Loggable, serge.geometry.Rectangle,

serge.common.EventAware

Represents an actor

addedToWorld(*world*)
Called when we are being added to the world

getAngle()
Return the angle for the actor

getLayerName()
Return our layer name

getNiceName()
Return a nice name for this actor

getPhysical()
Return the physical conditions

getSpriteName()
Return our sprite

init()
Initialize from serialized form

move(*x*, *y*)
Move by a certain amount

moveTo(*x*, *y*, *no_sync=False*, *override_lock=False*)
Move the center of this actor to the given location, unless it is locked

You can override the lock by passing True to override lock.

removedFromWorld(*world*)
Called when we are being removed from the world

renderTo(*renderer*, *interval*)
Render ourself to the given renderer

setAngle(*angle*, *sync_physical=False*, *override_lock=False*)
Set the angle for the visual

setLayerName(*name*)
Set the layer that we render to

setPhysical(*physical_conditions*)
Set the physical conditions

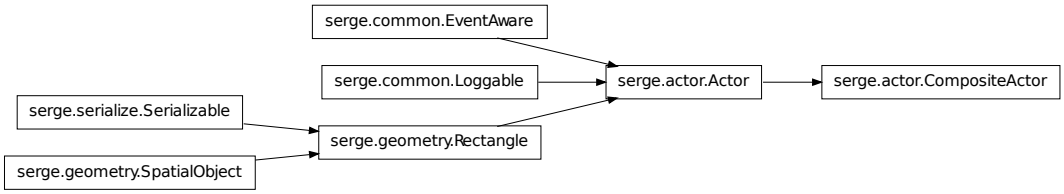
setSpriteName(*name*)
Set the sprite for this actor

setZoom(*zoom*)
Zoom in on this actor

syncPhysics(*spatial_only=False*)
Sync physics when the actors physical properties have been changed

updateActor(*interval*, *world*)
Update the actor status

5.2.2 CompositeActor



class serge.actor.**CompositeActor**(**args*, ***kw*)

Bases: serge.actor.Actor

An actor that can have children, which are also actors

World operations on the parent, like adding and removing, will also apply to the children.

If the children are removed from the parent then they are also removed from the world.

addChild(*actor*)
Add a child actor

addedToWorld(*world*)
Called when we are being added to the world

getChildren()
Return the list of children

getChildrenWithTag(*tag*)
Return all the children with a certain tag

hasChild(*actor*)
Return True if this actor already has this actor as a child

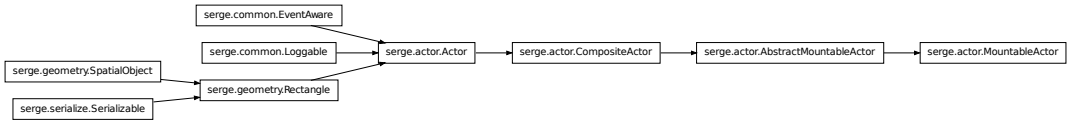
hasChildren()
Return True if this actor has children

removeChild(*actor*, *leave_in_world=False*)
Remove a child actor

removeChildren()
Remove all the children

removedFromWorld(*world*)
Called when we are being removed from the world

5.2.3 MountableActor



class serge.actor.**MountableActor**(**args*, ***kw*)
Bases: serge.actor.AbstractMountableActor

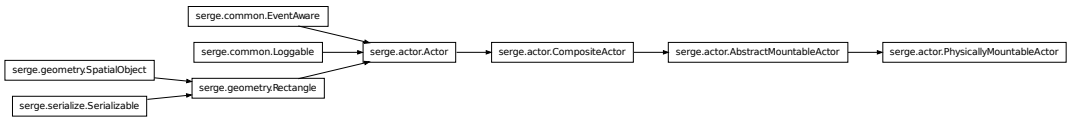
An actor that you can mount other actors to

The other actors are located at a certain position relative to the position of this actor. You can use this actor to create clusters either visually or functionally.

moveTo(*x*, *y*, *no_sync=False*, *override_lock=False*)
Move this actor

setAngle(*angle*, *sync_physical=False*, *override_lock=False*)
Set the angle for the visual

5.2.4 PhysicallyMountableActor



class serge.actor.**PhysicallyMountableActor**(*tag*, *name='‘*, *mass=0.0*, ***kw*)
Bases: serge.actor.AbstractMountableActor

An physical actor that you can mount other physical actors to

The other actors are located at a certain position relative to the position of this actor. You can use this actor to create clusters either visually or functionally.

All actors must be under the control of the physics engine.

addedToWorld(*world*)
The actor was added to the world

init()
Initialise from serialized form

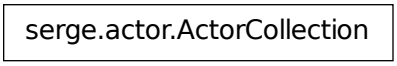
mountActor(*actor*, *(x, y)*, *original_rotation=False*)
Mount the actor with the given offset

moveTo(*x*, *y*, *no_sync=False*, *override_lock=False*)
Move this actor

setAngle(*angle*, *sync_physical=False*, *override_lock=False*)
Set the angle for the visual

unmountActor(*actor*)
Unmount the actor

5.2.5 ActorCollection



class serge.actor.**ActorCollection**
Bases: list

A list of actors

This class implements some useful methods which help to handle collections of actors.

findActorByName(*name*)
Return then actor with the given name

findActorsByTag(*tag*)
Return a collection of actors with the given tag

findActorsByTags(*tags*)
Return a collection of actors with at least one of the tags

forEach()
Returns an object suitable for mapping method calls to all the actors in the collection

Use this like, collection.forEach().setAngle(12)

hasActor(*actor*)
Return True if we have that actor

hasActorWithName(*name*)
Return True if the collection contains an actor with the given name

hasActorWithTag(*tag*)
Return True if the collection contains an actor with the given tag

numberOfActorsWithName(*name*)
Return the number of actors with the given name

numberOfActorsWithTag(*tag*)
Return the number of actors with the given tag

5.3 World

class serge.world.**World**(*name*)
Bases: serge.common.Loggable, serge.serialize.Serializable, serge.common.EventAware

The main world object

The *Engine* will control main worlds. Each world has a number of *Zones* which contain *Actors*.

activateWorld()
Called when the world is set as the current world

addActor(*actor*)
Add an actor to the world

addZone(*zone*)
Add a zone to the world

clearActors()
Clear all the actors

clearActorsExceptTags(*tags*)
Clear all actors except the ones with a tag in the list of tags

clearActorsWithTags(*tags*)
Clear all actors with a tag in the list of tags

clearZones()
Remove all the zones

deactivateWorld()
Called when the world is deactivated

findActorByName(*name*)
Return the actor with the give name in all zones

findActorsAt(*x*, *y*)
Return the actors at a certain location

findActorsByTag(*tag*)
Return all the actors in all zones based on the tag

getActors()
Return all the actors

getEngine()
Return the engine that we are owned by

hasActor(*actor*)
Return True if this actor is in the world

init()
Initialise from serialized state

processEvents(*events*)
Handle the events

removeActor(*actor*)
Remove the actor from the world

renderTo(*renderer*, *interval*)
Render all of our actors in active zones

rezoneActors()
Move actors to the right zone based on their spatial location

scheduleActorRemoval(*actor*)
Remove an actor at the end of the next update for the world

This method can be used to safely remove an actor from the world during the execution of the world update. It can sometimes be useful to do this when inside logic that is iterating over actors or inside the updateWorld event loop.

setEngine(*engine*)
Set the engine that we are owned by

setGlobalForce(*force*)
Set the global force for physics

setPhysicsStepsize(*interval*)
Set the maximum step size for physics calculations

setZoom(*zoom*, *x*, *y*)
Set the visual zoom on this world to zoom centered on x, y

sleepPhysicsForActors(*actors*)
Tell the actors to go to sleep from a physics perspective

The actors will still be visible and will still be updated but they will not update their physics. Useful for optimising when an actor does not need to interact with the physics simulation for a while.

If an actor is unzoned then this will have no impact on them

updateWorld(*interval*)
Update the objects in the world

wakePhysicsForActors(*actors*)
Tell the actors to go to wake up from a physics perspective

Actors that were put to sleep (via sleepPhysicsForActors) will be woken up and take part in the physics simulation again.

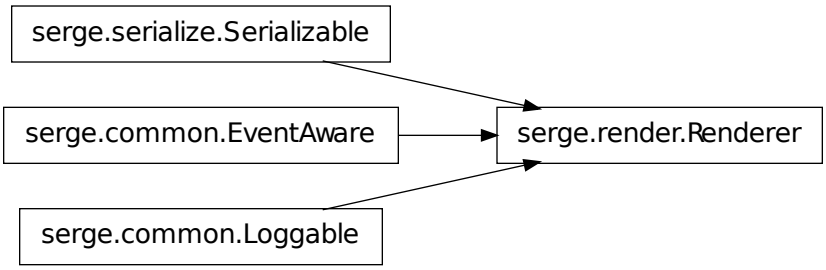
CHAPTER

SIX

RENDERERING

Contents

6.1 Renderer



class `serge.render.`**`Renderer`**(*width=640*, *height=480*, *title='Serge'*, *backcolour=(0, 0, 0)*, *icon=None*, *fullscreen=False*)

Bases: `serge.common.Loggable`, `serge.serialize.Serializable`, `serge.common.EventAware`

The main rendering component

addLayer(*layer*)
Add a layer to the rendering

clearLayers()
Clear all the layers

clearSurface()
Clear the surface

getCamera()
Return our camera

getLayer(*name*)
Return the named layer

getLayerBefore(*layer*)
Return the layer before the specified one in terms of rendering order

getLayers()
Return all the layers

getScreenSize()
Returns the screen size

getSurface()
Return the overall surface

init()
Initialise from serialized state

orderActors(*actors*)
Return the list of actors sorted by who should be processed first to correctly render

The actors are checked to see which layer they reside on and then this is used to order the returned list.

preRender()
Prepare for new rendering

removeLayer(*layer*)
Remove the layer from the rendering

removeLayerNamed(*name*)
Remove the layer with the specific name

render()
Render all the layers

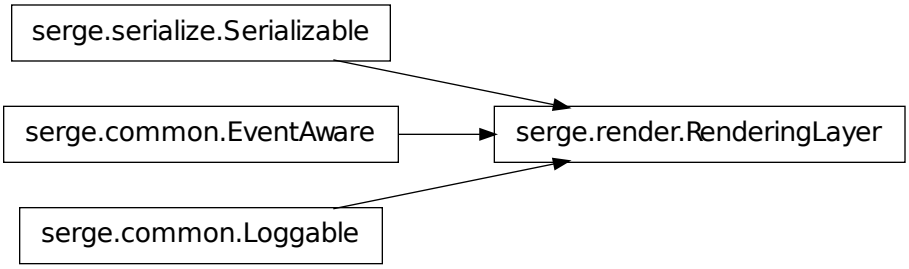
resetSurfaces()
Recreate the surfaces for our layers

When layers are added we sometimes need to reset the layers, for instance, virtual layers need to be shifted around so that they have the right order.

setCamera(*camera*)
Set our camera

6.2 Layers

6.2.1 RenderingLayer



class serge.render.**RenderingLayer**(*name*, *order*)

Bases: serge.common.Loggable, serge.serialize.Serializable, serge.common.EventAware

A layer on which to render things

This is the abstract version of the layer. Create subclasses of this to do useful things.

clearSurface()
Clear our surface

getNiceName()
Return the nice name for this layer

getSurface()
Return the surface

init()
Initialise from serialized state

initSurface(*renderer*)
Create the surface that we need to draw on

postRender()
Called after the layer has has had everything rendered on it

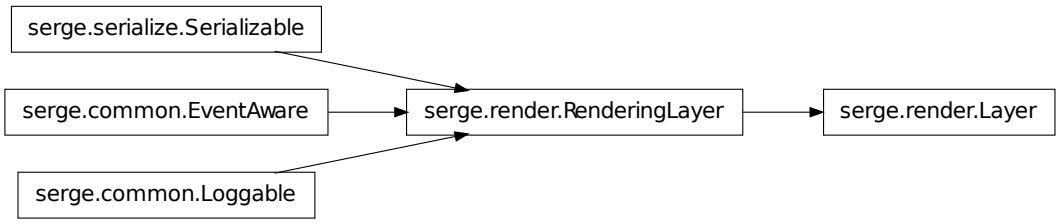
preRender()
Called before the layer has anything rendered to

render(*surface*)
Render to a surface

setStatic(*static*)
Determine whether this layer is static with respect to camera movements or not

setSurface(*surface*)
Set our surface

6.2.2 Layer



class serge.render.**Layer**(*name*, *order*)
Bases: serge.render.RenderingLayer

A rendering layer with its own surface

This type of layer is useful for compositing because you can do things to this layer once it has been rendered (eg shadows, glows, blurs etc).

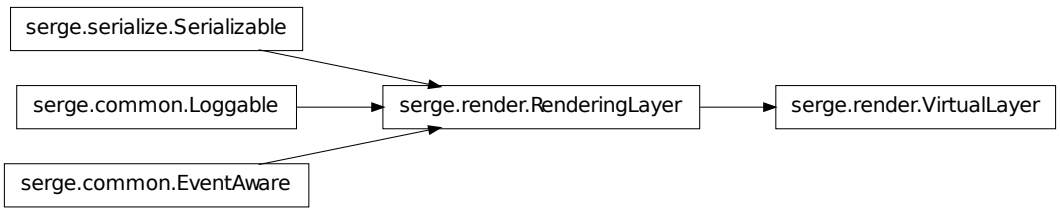
clearSurface()
Clear our surface

initSurface(*renderer*)
Create the surface that we need to draw on

We create a surface that is identical to the background for the main renderer.

render(*surface*)
Render to a surface

6.2.3 VirtualLayer



class serge.render.**VirtualLayer**(*name*, *order*)
Bases: serge.render.RenderingLayer

A rendering layer that doesn't have its own surface

This layer will render to the layer immediately before it in the rendering cycle.

clearSurface()
Clear our surface

Nothing to do here - handled by the real owner of the surface.

initSurface(*renderer*)
Create the surface that we need to draw on

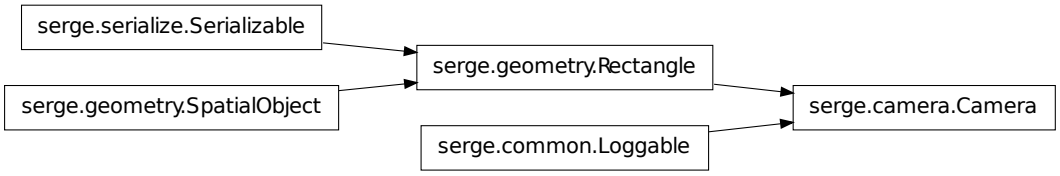
We do not want a surface ourself but we need the next surface in line as far as the renderer is concerned.

render(*surface*)
Render to a surface

Nothing to do here - handled by the real owner of the surface.

6.3 Cameras

6.3.1 Camera



class serge.camera.**Camera**
Bases: serge.common.Loggable, serge.geometry.Rectangle

Represents a camera

canSee(*actor*)
Return True if we can see the actor

canSeeActors(*actors*)
Return the actors that we can see from a list of actors

getRelativeLocation(*other*)
Return the relative location of one from another

getTarget()
Return the camera's target location

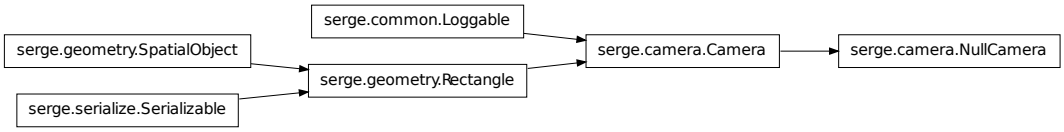
init()
Initialise from serialized

setTarget(*target*)
Set the target for the camera to head towards

setZoom(*zoom*, *x*, *y*)
Set the new zoom centered on the given x and y

update(*interval*)
Update the location of the camera

6.3.2 NullCamera



class serge.camera.**NullCamera**

Bases: serge.camera.Camera

A camera that can see everything

canSee(*actor*)

Can we see it? Yes we can

init()

Initialise

CHAPTER

SEVEN

VISUAL

Contents

7.1 Sprites

`serge.visual.`**`Sprites`**
Registry of all sprites (is a Store)

7.2 Fonts

`serge.visual.`**`Fonts`**
Registry of all fonts (is a FontStore)

7.3 Drawing

class `serge.visual.`**`Drawing`**
Bases: `object`

Represents something to draw on the screen

`flipHorizontal``()`
Flip the drawing horizontally

`flipVertical``()`
Flip the drawing vertically

getAngle()
Return the current angle

getCopy()
Return a copy

renderTo(*milliseconds*, *surface*, *(x, y)*)
Render to a surface

rotateBy(*angle*)
Rotate by a certain amount

scaleBy(*factor*)
Scale the image by a factor

setAlpha(*alpha*)
Set the overall alpha

setAngle(*angle*)
Set the rotation to a certain angle

setHorizontalFlip(*flip*)
Set the horizontal flip state

setScale(*scale*)
Set the scaling to a certain factor

setSize(*width*, *height*)
Set the size of the drawing directly

setVerticalFlip(*flip*)
Set the vertical flip state

7.4 SurfaceDrawing

class serge.visual.**SurfaceDrawing**(*width*, *height*)
Bases: serge.visual.Drawing

A visual object that renders to a surface.

You can create an instance of this class and then write to its surface or use this as a base class for your own class that will write the surface.

clearSurface()
Clear the surface

getSurface()
Return our surface

renderTo(*milliseconds*, *surface*, *(x, y)*)
Render to a surface

scaleBy(*factor*)
Scale the image by a factor

setAngle(*angle*)
Change the angle - returning the amount by which the sprite has shifted

setSize(*width*, *height*)
Set the size of the drawing directly

setSurface(*surface*)
Update our surface

7.5 Sprite

class serge.visual.**Sprite**
Bases: serge.visual.Drawing

An object that gets drawn on the screen

flipHorizontal()
Flip the drawing horizontally

flipVertical()
Flip the drawing vertically

getCell()
Return the current cell number

getCopy()
Return a copy of this sprite

getNumberOfCells()
Return the number of animation cells

getSurface()
Return the current surface

renderTo(*milliseconds*, *surface*, *(x, y)*)
Render to a surface

resetAnimation(*running*)
Reset the animation to the begining

scaleBy(*factor*)
Scale the image by a factor

setAlpha(*alpha*)
Set the overall alpha

setAngle(*angle*)
Change the angle - returning the amount by which the sprite has shifted

setCell(*number*)
Set the current cell number

setCells()
Create the cells for the animation of this sprite

setImage(*image*, *(width, height)*, *framerate=0*, *running=False*, *loop=True*, *one_direction=False*)
Set the image of this sprite

setSize(*width*, *height*)
Set the size of the drawing directly

7.6 FontStore

class serge.visual.**FontStore**
Bases: serge.registry.GeneralStore

A store for fonts

clearItems()
Clear the items

Leaves the DEFAULT item in there if it is there.

registerItem(*name*, *path*)
Register a font

7.7 Store

class serge.visual.**Store**
Bases: serge.registry.GeneralStore

Stores sprites

registerFromFiles(*name*, *path*, *number*, *framerate=0*, *running=False*, *rectangular=True*, *angle=0.0*, *zoom=1.0*, *start=1*, *loop=True*, *one_direction=False*)
Register a multi cell sprite from a number of files

The path should be a string with a single numerical substitution. We will pass the numbers 1..number to this substitution to find the names of the files.

registerItemsFromPattern(*pattern*, *prefix=''*, *w=1*, *h=1*, *framerate=0*, *running=False*, *rectangular=True*, *angle=0.0*, *zoom=1.0*, *loop=True*, *one_direction=False*)
Register all items matching a certain regular expression

registerMultipleItems(*names*, *path*, *w*, *h=1*, *rectangular=True*, *angle=0.0*, *zoom=1.0*, *one_direction=False*)
Register a number of sprites from a single image

The image must be a horizontal row of sprites and you must provide a list of names the same size as the row of sprites. Each other sprites will be created.

7.8 Text

class serge.visual.**Text**(*text*, *colour*, *font_name='DEFAULT'*, *font_size=12*, *justify='center'*)
Bases: serge.visual.Drawing

Some text to display

renderTo(*milliseconds*, *surface*, *(x, y)*)
Render to a surface

scaleBy(*scale*)
Scale our sprite by a certain certain amount

setAlpha(*alpha*)
Set our alpha

setAngle(*angle*)
Rotate our sprite by a certain angle

setColour(*colour*)
Set the colour

setFontSize (*font_size*)
Set our font size

setJustify (*justify*)
Set the justification

setText (*text*)
Set our text

CHAPTER

EIGHT

SOUND

Contents

8.1 Sounds

serge.sound.**Sounds**
The registry of all sounds (is an AudioRegistry)

8.2 Music

serge.sound.**Music**
The registry of all music (is a MusicStore)

8.3 AudioRegistry

class serge.sound.**AudioRegistry**
Bases: serge.registry.GeneralStore, serge.common.EventAware

Registry for audio

isPaused()
Return True if we are paused

isPlaying()
Return True if we are playing

pause()
Pause all sounds

play(*name*, *loops=0*)
Play a sound

toggle()
Toggle whether music or sound is playing or not

unpause()
Unpause all sounds

update(*interval*)
Update the registry looking for events

8.4 SoundItem

class serge.sound.**SoundItem**(*path*)
Bases: object

Represents a sound item

fadeout(*time*)
Fadeout the sound

get_volume()
Get the volume

isPlaying()
Return True if we are playing

pause()
Pause the music

play(*loops=0*)
Play the music

set_volume(*volume*)
Set the volume

stop()
Stop the music

unpause()
Pause the music

8.5 MusicStore

class serge.sound.**MusicStore**
Bases: serge.sound.AudioRegistry

Stores music

fadeout(*time*)
Fadeout the currently playing track

getVolume()
Get the volume

isPlaying()
Return True if we are playing

isPlayingSong(*name*)
Return True if the named song is playing

pause()
Pause all sounds

play(*name*, *loops=0*)
Play a sound

setPlaylist(*item_list*)
Set a playlist

setVolume(*volume*)
Set the volume

unpause()
Unpause all sounds

update(*interval*)
Update the registry looking for events

8.6 MusicItem

class serge.sound.**MusicItem**(*path*)
Bases: object

Represents a music item

pause()
Pause the music

play(*loops=0*)
Play the music

stop()
Stop the music

unpause()
Pause the music

CHAPTER

NINE

INPUT

Contents

9.1 Constants

M_LEFT - left mouse button

M_RIGHT - right mouse button

M_MIDDLE - middle mouse button

M_WHEEL_UP - scrolling the mouse wheel up

M_WHEEL_DOWN - scrolling the mouse wheel down

9.2 Keyboard

class serge.input.**Keyboard**

Bases: serge.common.Loggable

Represents the state of the keyboard

areAnyClicked()
Is any button clicked?

areAnyDown()
Is any button depressed?

getClicked()
Return a list of the keys that are clicked

getTextEntered()
Return any text entered since the last call

isAltDown()
Return True if the alt key is down

isClicked(*key*)
Return True if the key has been clicked

isControlDown()
Return True if the control key is down

isDown(*key*)
Return True if the key is down

isShiftDown()
Return True if the shift key is down

isUp(*key*)
Return True if the key is up

update(*interval*)
Update the state of the keyboard

class serge.input.**KeyState**
Bases: object

Represents the state of keyboard keys

getCopy()
Return a new copy of the key states

getState(*key*)
Return the state of a specific key

setState(*key*, *state*)
Set the state for a key

9.3 Mouse

class serge.input.**Mouse**(*engine*)
Bases: object

Represents the state of the mouse

clearClick(*MouseStateType*)
Clear a click event

getActorEvents(*world*, *layers=None*)
Return the type of events for each actor that we have hit

The optional parameter layers can be a list of layers that we are interested in. Only actors on the given layers will be returned.

getActorsUnderMouse(*world*)
Return all the actors that the mouse is over

getScreenPos()
Return the pixel location relative to the screen and camera

getStaticScreenPos()
Return the pixel location relative to the screen and NOT camera

isClicked(*MouseStateType*)
Return True if the mouse button is pressed

isDown(*MouseStateType*)
Return True if the mouse button is down

isUp(*MouseStateType*)
Return True if the mouse button is up

update(*interval*)
Update our mouse states

class serge.input.**MouseState**
Bases: object

A structure that contains the states of our mouse buttons.

getCopy()
Return a copy of this state

getState(*StateType*)
Return True if the specified button is pressed

setState(*StateType*, *state*)
Set the state of a specific key

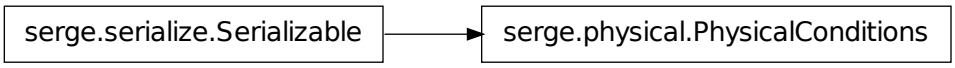
CHAPTER

TEN

PHYSICAL

Contents

10.1 PhysicalConditions



class serge.physical.**PhysicalConditions**(*mass=0.0*, *radius=0.0*, *velocity=(0.0, 0.0)*, *force=(0.0, 0.0)*, *width=0.0*, *height=0.0*, *fixed=False*, *friction=0.1*, *elasticity=1.0*, *group=0*, *layers=-1*, *update_angle=False*, *visual_size=False*)

Bases: serge.serialize.Serializable

Represents physical parameters of an object

This includes the mass, velocity, force applied, acceleration and the physical dimensions.

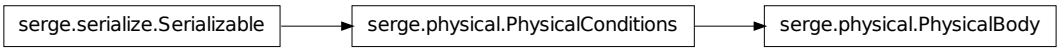
init()
Initialize from serialized form

setGeometry(*radius=None*, *width=None*, *height=None*)
Set the geometry

You must specify either the radius or the width and height

updateFrom(*physical_conditions*)
Update the properties and our physics object

10.2 PhysicalBody



class serge.physical.**PhysicalBody**(*mass*, ***kw*)

Bases: serge.physical.PhysicalConditions

Physical conditions for an infinitesimal object

The object has no dimensions (shape) but still has mass etc.

CHAPTER

ELEVEN

EVENTS

Contents

11.1 Broadcaster

class serge.events.**Broadcaster**

Bases: serge.common.EventAware

The main event broadcaster

11.2 getEventBroadcaster

events.**getEventBroadcaster**()

Returns the global event broadcaster which is used for global level events

11.3 Events

Here are the global event definitions:

```
# Occurs when one object collides with another
E_COLLISION = 'collision'

# Mouse events related to the left mouse button
#  - down is when the button is held down (fires continuously)
#  - up is when the button is released
#  - click is the mouse was down and then released
E_LEFT_MOUSE_DOWN = 'left-mouse-down'
E_LEFT_MOUSE_UP = 'left-mouse-up'
E_LEFT_CLICK = 'left-click'

# Mouse events related to the right mouse button
```

```
#  - down is when the button is held down (fires continuously)
#  - up is when the button is released
#  - click is the mouse was down and then released
E_RIGHT_MOUSE_DOWN = 'right-mouse-down'
E_RIGHT_MOUSE_UP = 'right-mouse-up'
E_RIGHT_CLICK = 'right-click'

# Mouse events related to the wheel
#  - wheel up the mouse wheel was moved up
#  - wheel down the mouse wheel was moved down
E_MOUSE_WHEEL_UP = 'wheel-up-click'
E_MOUSE_WHEEL_DOWN = 'wheel-down-click'

# Events related to actor and the world
E_ADDED_TO_WORLD = 'added-to-world'
E_REMOVED_FROM_WORLD = 'remove-from-world'

# Events related to the world or layers
#
# The world is activated when it
# becomes the current world for the engine.
# The previously activated world is deactivated.
#
# Before and after render are triggered relative
# to rendering the whole world or the layer
E_ACTIVATE_WORLD = 'activate-world'
E_DEACTIVATE_WORLD = 'deactivate-world'
E_BEFORE_RENDER = 'before-render'
E_AFTER_RENDER = 'after-render'

# Events related to the keyboard
E_KEY_DOWN = 'key-down'
E_KEY_UP = 'key-up'
E_KEY_CLICKED = 'key-clicked'

# Events related to the engine
E_BEFORE_STOP = 'before-stop'  # The stop method has been called and the engine is about to quit
E_AFTER_STOP = 'after-stop'  # The stop method has been called and the engine is quiting

# Events related to movement
E_ACTOR_ARRIVED = 'actor-arrived'

# Events related to sound and music
E_TRACK_ENDED = 'track-ended'

# Drag and drop events
E_DRAG_START = 'drag-start'
E_DRAG_ENDED = 'drag-ended'
E_DROPPED_ON = 'dropped-on'

# Events related to entering information
E_ACCEPT_ENTRY = 'accept-entry'
```

CHAPTER

TWELVE

GEOMETRY

Contents

12.1 SpatialObject

class serge.geometry.**SpatialObject**
Bases: object

Represents a spatial object

isInside(*other*)
Return True if this object is inside another

isOverlapping(*other*)
Return True if this object overlaps another

12.2 Rectangle

class serge.geometry.**Rectangle**(*x=0*, *y=0*, *w=0*, *h=0*)
Bases: serge.geometry.SpatialObject, serge.serialize.Serializable

Represents a rectangle

classmethod fromCenter(*cx*, *cy*, *w*, *h*)
Return a new rectangle giving the center x, y and width, height

getArea()
Return the area of the shape

getDistanceFrom(*other*)
Return the distance we are from another

getOrigin()
Get the left and top coords

getRelativeLocation(*other*)
Return the relative location of another object

getRelativeLocationCentered(*other*)
Return the relative location of another object

getSpatial()
Return spatial details

getSpatialCentered()
Return spatial details

init()
Initialize from serialized

isInside(*other*)
Return True if this object is inside another

isOverlapping(*other*)
Return True if this object overlaps another

move(*dx*, *dy*)
Move the actor

moveTo(*x*, *y*, *override_lock=False*)
Move the center of this object to the given location, unless it is locked

This is the main method used to implement the position of the shape. This is the one to override.

resizeBy(*w*, *h*)
Resize the spatial by the given extent

resizeTo(*w*, *h*)
Resize the spatial by the given extent

scale(*factor*)
Rescale the spatial extent

setOrigin(*x*, *y*)
Set the left and top coords

setSpatial(*x*, *y*, *w*, *h*)
Set the spatial details of ourself

setSpatialCentered(*x*, *y*, *w*, *h*)
Set the spatial details of ourself

12.3 Point

class serge.geometry.**Point**(*x*, *y*)
Bases: serge.geometry.Rectangle

Represents a point

isInside(*other*)
Return True if this object is inside another

isOverlapping(*other*)
Return True if this object overlaps another

12.4 SimpleRect

class serge.geometry.**SimpleRect**(**args*)
Bases: list

A simple rectangle implementation

collidepoint(*x*, *y*)
Return True if this rectangle collides with another

colliderect(*other*)
Return True if this rectangle collides with another

contains(*other*)
Return True if this rectangle contains another

inflate(*w*, *h*)
Inflate to new width and height staying in the same centered place

inflate_ip(*w*, *h*)
Inflate current rectangle to new width and height staying in the same centered place

move_ip(*dx*, *dy*)
Move in place

12.5 Vec2d

class serge.simplevecs.**Vec2d**(*x_or_pair=None*, *y=None*)
Bases: object

2d vector class, supports vector and scalar operators, and also provides some high level functions

angle
Gets or sets the angle (in radians) of a vector

angle_degrees
Gets or sets the angle (in degrees) of a vector

cpvrotate(*other*)
Uses complex multiplication to rotate this vector by the other.

cpvunrotate(*other*)
The inverse of cpvrotate

cross(*other*)
The cross product between the vector and other vector v1.cross(v2) -> v1.x*v2.y - v2.y*v1.x

Returns The cross product

dot(*other*)
The dot product between the vector and other vector v1.dot(v2) -> v1.x*v2.x + v1.y*v2.y

Returns The dot product

classmethod from_param(*arg*)
Used by ctypes to automatically create Vec2ds

get_angle_between(*other*)
Get the angle between the vector and the other in radians

Returns The angle

get_angle_degrees_between(*other*)
Get the angle between the vector and the other in degrees

Returns The angle (in degrees)

get_dist_sqrd(*other*)
The squared distance between the vector and other vector It is more efficent to use this method than to call get_distance() first and then do a sqrt() on the result.

Returns The squared distance

get_distance(*other*)
The distance between the vector and other vector

Returns The distance

get_length()
Get the length of the vector.

Returns The length

get_length_sqrd()
Get the squared length of the vector. It is more efficent to use this method instead of first call get_length() or access .length and then do a sqrt().

Returns The squared length

int_tuple
Return the x and y values of this vector as ints

length
Gets or sets the magnitude of the vector

normalize_return_length()
Normalize the vector and return its length before the normalization

Returns The length before the normalization

normalized()
Get a normalized copy of the vector Note: This function will return 0 if the length of the vector is 0.

Returns A normalized vector

static ones()
A vector where both x and y is 1

rotate(*angle_radians*)
Rotate the vector by angle_radians radians.

rotate_degrees(*angle_degrees*)
Rotate the vector by angle_degrees degrees.

rotated(*angle_radians*)
Create and return a new vector by rotating this vector by angle_radians radians.

Returns Rotade vector

rotated_degrees(*angle_degrees*)
Create and return a new vector by rotating this vector by angle_degrees degrees.

Returns Rotade vector

static unit()
A unit vector pointing up

static `zero`()
A vector of zero length

CHAPTER

THIRTEEN

COMMON

Contents

13.1 Loggable

class serge.common.**Loggable**
Bases: object

A helper class that adds a logger to a class

Each instance of the class will have a *log* attribute and can use this to log output. The *log* attrbute is a logger with the usual *debug*, *warn*, *info*, and *error* methods.

addLogger()
Add a logger

13.2 getLogger

registry.**getLogger**(*name*)
Return a named logger - mainly used when logging from a non-loggable object

13.3 EventAware

class serge.common.**EventAware**
Bases: object

A mixin class that allows objects to respond to events

handleEvent(*event*)
Handle an incoming event

initEvents()
Initialise the events system

linkEvent(*name*, *callback*, *arg=None*)
Link an event to a callback

processEvent(*event*)
Process an incoming event

registerEvent(*event*)
Register an event

registerEvents(*events*)
Register a number of events

registerEventsFromModule(*module*)
Register all events found in the module

Events must be strings and their name must be of the form E_THE_NAME

ie: Begins with an 'E' and is all uppercase

unlinkEvent(*name*, *callback=None*)
Unlink an event from a callback

13.4 Serializable

class serge.serialize.**Serializable**
Bases: object

A mixing class to help serialize and deserialize objects

asString()
Return the properties of this object as a string

copy()
Return another copy of this item

classmethod createInstance()
Return an instance of the class with all default properties set

classmethod fromFile(*filename*)
Return a new instance from a file

classmethod fromString(*text*)
Return a new instance from a string

init()
Implement this method to do any object initialization after unpickling

toFile(*filename*)
Store this object in a file

13.5 GeneralStore

class serge.registry.**GeneralStore**
Bases: serge.serialize.Serializable

Stores things

clearItems()
Clear all the items

duplicateItem(*name*, *new_name*)
Create a duplicate of the named item with a new name

getItem(*name*)
Return an item

getItemDefinitions()
Return all the item definitions

getItems()
Return all the items

getNames()
Return the names of all the items

init()
Initialise from serialized form

registerItem(*name*, **args*, ***kw*)
Register an item

removeItem(*name*)
Remove the named item

setPath(*path*)
Set our base path to locate images

CHAPTER

FOURTEEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z