Game Engine Programming Guide

D. Playne & A. Leist

School of Natural and Computational Sciences

Massey University – Albany

Modified by N. Baghaei

2020

Chapter 1

Introduction

Some of the course examples, lab exercises and assignments in 159.261 make use of this game engine. This guide provides a simple introduction and explanation of how the game engine works and serves to complement the lectures with a reference.

Game Engine

The game engine is implemented in the Java programming language in a file named GameEngine.java and takes care of a number of the more complicated parts of writing a game. It also hides some of the object-oriented features of Java from you. You should note that this is not a good example of how to write a program in Java. It has been written this way to ease your transition from the introductory courses 159.10X that use the imperative programming language C. This section will introduce you to how to use the game engine with a simple example. If you are already familiar with Java, you are encouraged to look through the game engine source code.

The game engine takes care of creating and handling a window that appears on the screen as well as handling user input which can be optionally used by games. The engine can be used by simply creating a Java class (the name of the file must match the name of the class) that extends the game engine, this uses an object-oriented feature called inheritance. The game class should have a main function that calls the function createGame. The code to do this should be almost identical for every program based on the game engine (only the name of the class and file will change). An example of how to do this is shown in Listing 1.1.

Listing 1.1: Example showing a game called SimpleGame that uses GameEngine (must be written in a file named SimpleGame.java).

```
public class SimpleGame extends GameEngine {
    public static void main (String args[ ] ) {

        createGame (new SimpleGame ( ));

    }

     . . .

}
```

When the game is first created, the GameEngine will call a function called init that the game can use to initialise any variables for the game, set the size of the window etc. In our SimpleGame example there are no variables for the game so the init function simply sets the size of the window to 500x500 - the size of the window can be changed by calling the GameEngine function setWindowSize(width,height). The init function for SimpleGame is shown in Listing 1.2

The GameEngine has an internal timer that will automatically call two functions that8 must be imple-mented by any game - update and paintComponent. The update function is responsible for the dynamics of the game, this is where any changes to the state of the game should be written. The paintComponent method is responsible for displaying the game on the screen. The GameEngine calls both of these functions multiple times per second based on the update rate or framerate of the game (usually 30 frames per second). A simple diagram of the flow of control for a game written using GameEngine can be seen in Figure 1.1.

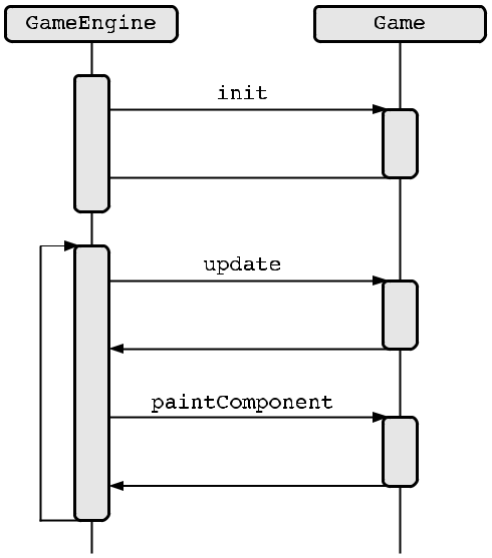


Figure 1.1: Flow control of a game based on GameEngine.

In this simple example there is no game state and there are no dynamics (it is not really a game). The update function does nothing but the paintComponent will use several functions from the GameEngine to draw the game on the screen. In this case it simple draws a black background and then a white circle on the screen. The code for this is shown in Listing 1.2 and will produce the output shown in Figure 1.2.

Listing 1.2: Initialisation function for SimpleGame.

```
public class SimpleGame extends GameEngine {

    public static void main (String args [ ] ) {

        createGame (new SimpleGame( ));

    }

    public void init ( ) {

        setWindowSize (500, 500);

    }
```

```
        public void update (double dt)  {
        }
        public void paintComponent ( )  {
                changeBackgroundColor (black);
                clearBackground (500, 500);
                changeColor (white);
                drawCircle (1 0 0, 100, 50);
        }
}
```

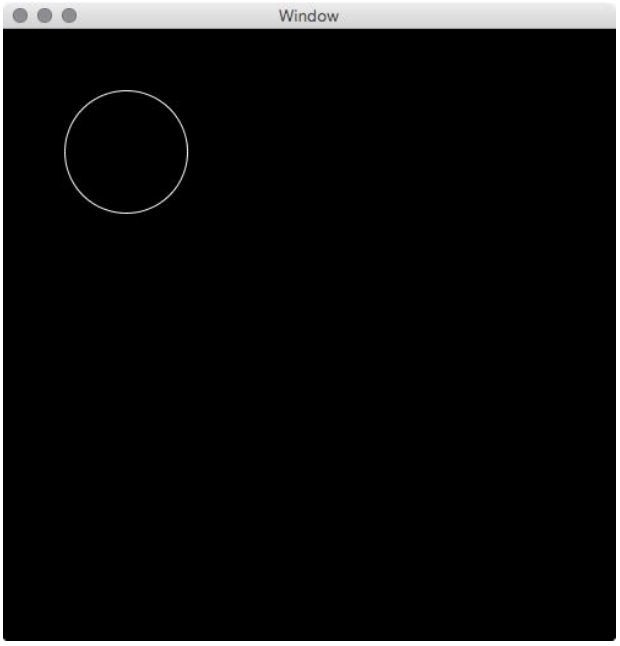


Figure 1.2: Output on the screen of the SimpleGame example.

Declaring Variables

There are no real global variables in Java, but games written using this game engine can declare variables that will be accessible from any function in the game. Variables should be declared outside the functions (but still within the enclosing class) to make them available throughout the program. While the variables are declared outside functions, they should be assigned values in the init function.

Initialisation Function

The init function is where any initialisation code should be performed. This function will only be called once before the game starts to run and is responsible for assigning values to all the variables in the game. Common operations performed in the init function include: initialising variables to default or initial values, allocating memory for arrays, loading images from file and loading audio clips.

Update Function

The update function is where the main logic of the game is implemented. This function will be called many times per second (dependent on the framerate of the game) and receives a single parameter dt that tells the function how much time has passed since the game was last updated (dt is measured as a fraction of a second). The dt parameter can be used to ensure that the game objects move at the same rate irrespective of the actual framerate the game manages to achieve.

Common operations performed inside the update function include moving game objects, checking for boundary conditions, checking for collisions between other objects etc.

PaintComponent Function

The paintComponent function is where all of the code that draws objects on the screen belongs. This function should not try to move or update game objects and should only draw them on the screen. Code in this function will make heavy use of all of the drawing functions described in the following chapter.

Input Functions

The game engine supports a number of different input functions for receiving input from the mouse and/or the keyboard. These include pressing/releasing keys on the keyboard and mouse buttons as well as moving the mouse around on the screen. Use of these input functions allow the user to interact with the game and control certain game objects. Care should be taken that the logic inside an input function does not interfere with the game logic in the update function.

Summary

This game engine provides a simple framework for developing 2D games with Java by providing some common functionality and abstracting away some unnecessary syntax. The main points to remember when writing a game using this game engine is where different code belongs. The previous sections describe where to declare variables, initialise variables, allocate memory, load images/audio files, update the game, draw objects on the screen and receive input from the user. The following section of this guide provides a detailed description of the functions the game engine provides and in some cases the common usage of the functions.

Chapter 2

Game Engine Functions

The following sections describe the major functions that the Java game engine provides for you to use.

If you look through the game engine source code, you will discover that there are additional functions that relate to the internal workings of the game engine.

2.1 Window Functions

void setWindowSize(int width, int height)

Sets the useable size of the game window to (width, height). This function will automatically add the extra size need for the border around the window.

---

int width()

Returns the current width of the useable area of the game window. Allows games to avoid using hardcoded sizes that prevent the game from being scaled to different sizes.

---

int height()

Returns the current height of the useable area of the game window. Allows games to avoid using hardcoded sizes that prevent the game from being scaled to different sizes.

2.2 Drawing Functions

The following functions are used to draw the background and primitive shapes on the game engine frame.

See the following Image Function section for details on loading and drawing images.

---

void changeBackgroundColor(Color c)

Usage: paintComponent()

Sets the colour that the game engine will use to draw the background. Takes one parameter c that must be one of the following pre-defined colours {black, orange, pink, red, purple, blue, green, yellow, white}. Other colours can be used by calling the following method void changeBackgroundColor(int red, int green, int blue) that allows a colour defined with rgb values.

---

void changeBackgroundColor(int red, int green, int blue)

Usage: paintComponent()

Sets the colour that the game engine will use to draw the background. Takes three parameters red, green, blue that are integers in the range [0, 255] that represent a colour in RGB space.

---

void clearBackground(int width, int height)

Usage: paintComponent()

Clears the rectangle (0,0) to (width,height) to the current background colour of the game engine. This function is usually called at the beginning of the paintComponent() with the width and height of the entire game panel. This will clear the entire screen to the background colour, the game graphics can then be drawn over the top of this background.

---

void changeColor(Color c)

Usage: paintComponent()

Sets the current drawing colour of the game engine. Any shape or text drawn following this call will be drawn with the colour c (at least until the drawing colour is changed again). The parameter c must be one of the following pre-defined colours {black, orange, pink, red, purple, blue, green, yellow, white}. Other colours can be used by calling the following method void changeColor(int red, int green, int blue) that allows a colour defined with rgb values.

---

void changeColor(int red, int green, int blue)

Usage: paintComponent()

Sets the current drawing colour of the game engine. Takes three parameters red, green, blue that are integers in the range [0, 255] that represent a colour in RGB space.

---

void drawLine(double x1, double y1, double x2, double y2)

Usage: paintComponent()

Draws a line with the graphics context from the point (x1,y1) to (x2, y2). The current drawing colour of the game engine will determine the colour of the line drawn.

void drawLine(double x1, double y1, double x2, double y2, double l)

Usage: paintComponent()

Draws a line with the graphics context from the point (x1,y1) to (x2, y2). The width of the line is specified by the parameter l. The current drawing colour of the game engine will determine the colour of the line drawn.

---

void drawRectangle(double x, double y, double w, double h)

Usage: paintComponent()

Draws a rectangle (only the outside lines) on the graphics context from the point (x,y) with a width of w and a height of h. The current drawing colour of the game engine will determine the colour of the rectangle drawn.

---

void drawRectangle(double x, double y, double w, double h, double l)

Usage: paintComponent()

Draws a rectangle (only the outside lines) on the graphics context from the point (x,y) with a width of w and a height of h. The width of the line is specified by the parameter l. The current drawing colour of the game engine will determine the colour of the rectangle drawn.

---

void drawSolidRectangle(double x, double y, double w, double h)

Usage: paintComponent()

Draws a solid or filled in rectangle on the graphics context from the point (x,y) with a width of w and a height of h. The current drawing colour of the game engine will determine the colour of the rectangle drawn.

---

void drawCircle(double x, double y, double radius)

Usage: paintComponent()

Draws a circle (only the outside line) on the graphics context centred at the point (x,y) with a radius of radius. The current drawing colour of the game engine will determine the colour of the circle drawn.

---

void drawCircle(double x, double y, double radius, double l)

Usage: paintComponent()

Draws a circle (only the outside line) on the graphics context centred at the point (x,y) with a radius of radius. The width of the line is specified by the parameter l. The current drawing colour of the game engine will determine the colour of the circle drawn.

---

void drawSolidCircle(double x, double y, double radius)

Usage: paintComponent()

Draws a solid or filled in circle on the graphics context centred at the point (x,y) with a radius of radius.

The current drawing colour of the game engine will determine the colour of the circle drawn.

---

void drawText(double x, double y, String s)

Usage: paintComponent()

Draw the text in the string s on the graphics context with the bottom-left corner of the text at point (x,y). By default this function will draw the text in Arial font with a font size of 40. The current drawing colour of the game engine will determine the colour of the text drawn.

---

void drawBoldText(double x, double y, String s)

Usage: paintComponent()

Draw the text in the string s on the graphics context with the bottom-left corner of the text at point (x,y). By default this function will draw the text in Arial bold font with a font size of 40. The current drawing colour of the game engine will determine the colour of the text drawn.

---

void drawText(double x, double y, String s, String font, int size)

Usage: paintComponent()

Draw the text in the string s on the graphics context with the bottom-left corner of the text at point (x,y). The font used to draw this text is defined by the parameter font and the font-size by size. The font specified must be supported by your system. The current drawing colour of the game engine will determine the colour of the text drawn.

---

void drawBoldText(double x, double y, String s, String font, int size)

Usage: paintComponent()

Draw the text in the string s on the graphics context with the bottom-left corner of the text at point (x,y). The font used to draw this text will be bold and is defined by the parameter font and the

font-size by size. The font specified must be supported by your system. The current drawing colour of the game engine will determine the colour of the text drawn.

2.3 Image Functions

The following functions can be used to load images from file, extract sub-images from a previously loaded image and draw images on the screen. These functions should be used to load images from file (usually in the init function) and saved in an Image variable. These images can then be drawn on the screen to represent various game objects using the drawImage functions.

Image loadImage(String filename)

Open the file specified by the parameter filename and load the image contained within it. The image will be returned as an Image variable that must be stored and can be drawn on the screen later using drawImage.

Image subImage(Image image, int x, int y, int w, int h)

Extract a subsection of image from (x,y) with size (w,h) and return it as a separate image. The coordinates x,y and size w,h are defined in pixels. This function is usually used to extract individual sprites from a sprite-sheet.

---

void drawImage(Image image, double x, double y)

Usage: paintComponent()

Draws the image defined by image on the graphics context with the top-left corner at (x,y). The image will be drawn with the native size (in pixels) of the image. To draw the image larger/smaller than its original size use the following function void drawImage(Image image, double x, double y, double w, double h).

---

void drawImage(Image image, double x, double y, double w, double h)

Usage: paintComponent()

Draw the image defined by image on the graphics context with the top-left corner at (x,y). The image will be drawn with a width of w and a height of h. Java will deal with scaling the image to the desired size.

2.4 Transform Functions

The transform functions in GameEngine can transform the graphics context used to draw two-dimensional objects on the screen. These transforms are all affine transforms - they are linear transforms that preserve points and straight lines. Any set of lines that are parallel before an affine transform will remain parallel after it is applied. The supported transformations are - translate, rotate, scale and shear.

In addition to these transformations there are two functions for saving and restoring transforms. The saveCurrentTransform will save the current transformation of the graphics context which can be re-stored later by calling the function restoreLastTransform. These transforms are stored on a stack which is a first-in last-out data structure. This system allows multiple transforms to be saved, calling restoreLastTransform will restore the saved transforms in the reverse order they were added. It is im-portant to note that all transforms will be reset between frames. The transform will be reset to a default before each call to paintComponent.

---

void saveCurrentTransform()

Usage: paintComponent()

Saves the current transform of the graphics context and adds it to the stack of saved transforms. General usage is to save the current transform before drawing an object and then restoring it afterwards.

---

void restoreLastTransform()

Usage: paintComponent()

Restores the most recently saved transform of the graphics context and removes it from the stack. Repeat-edly calling this function will cycle the graphics context through all of the saved transforms in the reverse order they were saved in.

---

void translate(double x, double y)

Usage: paintComponent()

Applies a translation to the graphics context. This transform moves the graphics context by (x,y) pixels in the x- and y-dimensions respectively. This translation will occur within the coordinate space of the existing graphics context, previous transformations will have an effect on the transformation. An example of this transform is shown in Figure 2.1.

---

void rotate(double a)

Usage: paintComponent()

Applies a rotation to the graphics context. The rotation transformation will rotate the graphics context by

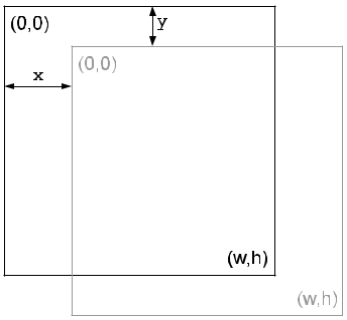


Figure 2.1: Example of a translation transformation.

a degrees around the origin (0,0). An example of how this transformation works can be seen in Figure 2.2.

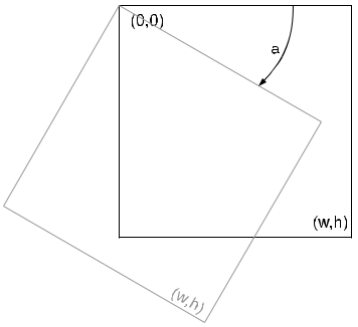


Figure 2.2: Example of a rotation transformation.

---

void scale()

Usage: paintComponent()

Applies a scale transformation to the graphics context. This will scale the graphics context by a factor of (x,y) in the x- and y-dimensions respectively. Figure 2.3 shows an example of the scale transform.

---

void shear()

Usage: paintComponent()

Applies a shear transformation to the graphics context. This will apply a shear of (x,y) in the x- and y-dimensions respectively. An example of this shear transform is shown in Figure 2.4.

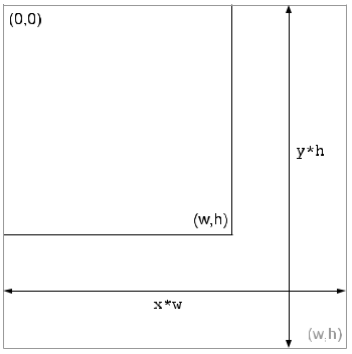


Figure 2.3: Example of a scale transformation.

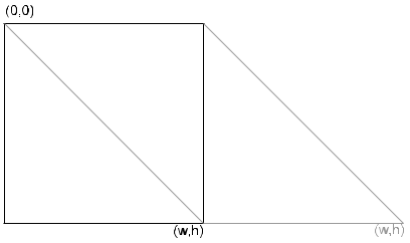


Figure 2.4: Example of a shear transformation.

Common Transform Usage

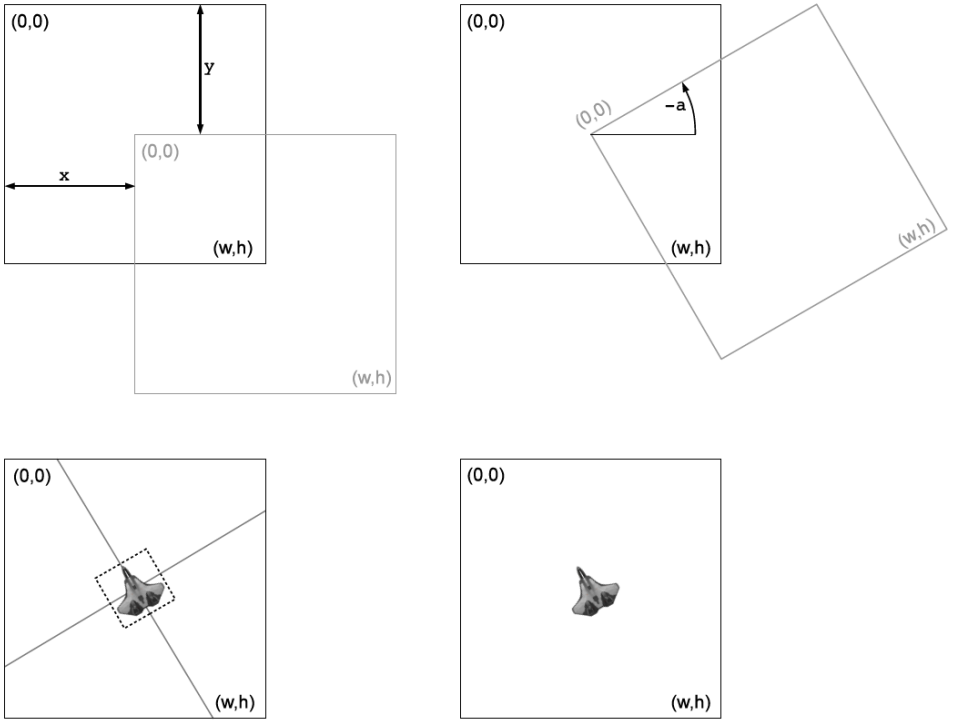
The most common usage of graphics transforms in games is for drawing objects with a given position and rotation. For example, drawing the spaceship from the Lab exercises. An important part process is to save the current transform before drawing and to restore it at the end of the process. This ensures that the transforms used to draw one object do not affect any others. After the transform is saved, a translation is applied to move the graphics context to the centre of where the object should be drawn. Then a rotation is applied to rotate the context to the desired angle, the rotation is always applied around the point (0,0). Finally, the object is drawn centred around the point (0,0). For example, if an image is 60x60 pixels, it will be drawn at the point (-30, -30) to (30, 30). The process for drawing an object at the position (x,y) with a rotation a is shown in Figure 2.5.

2.5 Input Functions

The input functions in the game engine are called automatically whenever an input event occurs. There are a whole set of different input events that can occur and will cause the matching input function to be called. For example, whenever the user presses a key down on the keyboard, the

game engine will call the keyPressed(KeyEvent event) function. If you have not written an implementation of this function in your game, nothing will happen. If you have written an implementation then the function will be called

Figure 2.5: Transforms used to draw an object centred at position (x,y) with rotation a. First the current transform is saved (not shown), the graphics context is translated to position (x,y) (top-left), the context is rotated by angle a (top-right), the object is drawn (bottom-left) and finally the saved transform is restored (bottom-right). Sprites were made by MillionthVector (millionthvector.blogspot.co.nz) and are licensed under a Creative Commons Attribution 4.0 International License.



and your game can respond to input.

The input functions will receive a single parameter which is either a KeyEvent for keyboard functions or a MouseEvent for mouse functions. These parameters can be used to determine information about the event that occurred such as the position of the mouse or which key was actually pressed. Each of the functions are described in more detail as follows.

---

void keyPressed(KeyEvent event)

Usage: Called automatically whenever the user presses a key on the keyboard. See the following section KeyEvent for details on how to extract information from a keyboard event.

---

void keyReleased(KeyEvent event)

Usage: Called automatically whenever the user releases a key on the keyboard. See the following section KeyEvent for details on how to extract information from a keyboard event.

---

void keyTyped(KeyEvent event)

Called automatically whenever the user presses and immediately releases a key on the keyboard. See the following section KeyEvent for details on how to extract information from a keyboard event.

KeyEvent event

The KeyEvent parameter sent to the keyboard events contains information about the keyboard event that has just occurred. The main information needed from the event is which key was pressed/re-leased/typed. For the keyPressed and keyReleased events, the information can be obtained by calling event.getKeyCode(). Calling this function will return an integer key code representing the key that can then be compared to a set of key codes to determine which key it represents. Some examples of key codes are the up-arrow KeyEvent.VK_UP, the spacebar KeyEvent.VK_SPACE and the enter key KeyEvent.VK_ENTER.

The keyTyped is higher-level function and uses the function event.getKeyChar() to extract the informa-tion of which key was pressed. Key typed events are not generated for keys that don't generate Unicode characters (action keys, modifier keys, etc).

Common Keyboard Function Usage

The following code segment (Listing: 2.1) shows an example of a keyboard input function that prints a message whenever the spacebar or enter key is pressed.

Listing 2.1: Example use of the keyPressed function and the KeyEvent.

```
public void k e y P r e s s e d ( KeyEvent  e v e n t )  {
     i f ( e v e n t . getKeyCode ( ) == KeyEvent .VK_SPACE)  {
          System . out . p r i n t l n              Bar" ) ;
          (“Space
     }  else  i f ( e v e n t . getKeyCode ( )   == KeyEvent .VK_ENTER)  {
          System . out . println ( “Enter     Key" ) ;
     }
}
```

---

void mousePressed(MouseEvent event)

Called automatically whenever the user presses a mouse button. See the following section MouseEvent for details on how to extract information from a mouse event.

---

void mouseReleased(MouseEvent event)

Called automatically whenever the user releases a mouse button. See the following section MouseEvent for details on how to extract information from a mouse event.

---

void mouseClicked(MouseEvent event)

Called automatically whenever the user presses and immediately releases a mouse button. See the following section MouseEvent for details on how to extract information from a mouse event.

---

void mouseEntered(MouseEvent event)

Called automatically whenever the mouse cursor enters the game window. See the following section MouseEvent for details on how to extract information from a mouse event.

---

void mouseExited(MouseEvent event)

Called automatically whenever the mouse cursor exits the game window. See the following section MouseEvent for details on how to extract information from a mouse event.

---

void mouseMoved(MouseEvent event)

Called automatically whenever the user moves the mouse (inside the game window). See the following section MouseEvent for details on how to extract information from a mouse event.

---

void mouseDragged(MouseEvent event)

Called automatically whenever the user moves the mouse (inside the game window) while holding down a mouse button. See the following section MouseEvent for details on how to extract information from a mouse event.

MouseEvent event

The Mouse parameter sent to the mouse event functions contains information about the mouse event that has just occurred. The main information needed from the event is which mouse button was pressed/released (if any) and where the mouse is. The mouse button pressed can be obtained by calling event.getButton() which will return an integer code for the mouse button. This can be compared to the values MouseEvent.BUTTON1, MouseEvent.BUTTON2 and MouseEvent.BUTTON3 to determine which mouse button was used in the event.

The current position of the mouse cursor (in the game window) can be determined by calling the functions event.getX() and event.getY() that return the x and y coordinates of the cursor respectively. An example of a common mouse event function is shown in the following section.

Common Mouse Function Usage

The following code segment (Listing: 2.2) shows an example of a mouse input function that prints out which mouse button was just pressed and where the mouse cursor is on the window.

Listing 2.2: Example use of the mousePressed function and the MouseEvent.

```
public void mousePressed ( MouseEvent  e v e n t ) {
    i f ( e v e n t . getButton ( ) == MouseEvent .BUTTON1) {
        System . out . p r i n t l n ( "Mouse  Button  1 p r e s s e d " ) ;
    }  else    i f ( e v e n t . getButton ( ) == MouseEvent .BUTTON2) {
        System . out . p r i n t l n ( "Mouse  Button  2 p r e s s e d " ) ;
    }  else    i f ( e v e n t . getButton ( ) == MouseEvent .BUTTON3) {
    }    System . out . p r i n t l n ( "Mouse  Button  3 p r e s s e d " ) ;

    int x = e v e n t . getX ( ) ;
    int y = e v e n t . getY ( ) ;
    System . out . p r i n t l n ( "a t  p o s i t i o n ( "+x  + " , " + y +      ")");

}
```

2.6 Audio Functions

The game engine provides a set of functions for loading and playing audio files. The game engine is limited in the type and size of the audio files it can play. The engine should support .wav files. Audio files can be played with two main functions - playAudio and startAudioLoop. The function playAudio will play an audio clip once after which it will stop. This function can be called many times and have many version of the same audio clip playing at once. The startAudioLoop function will start the audio clip playing on a continuous loop, only one loop of an audio clip can be playing at once. These (and additional) audio functions are described below:

AudioClip loadAudio(String filename)

Opens the file specified by filename and loads the audio clip information from it. Returns an AudioClip that can be stored and used to play the audio later.

---

void playAudio(AudioClip audioClip)

Plays the audio clip specified by the parameter audioClip with a default volume setting. The clip will be played once and will then stop.

---

void playAudio(AudioClip audioClip, float volume)

Plays the audio clip specified by the parameter audioClip with a volume specified by the parameter volume. Note that the volume is specified in decibels. The clip will be played once and will then stop.

---

void startAudioLoop(AudioClip audioClip)

Starts the audio clip specified by the parameter audioClip playing on a continuous loop. The clip can be stopped by calling the function stopAudioLoop and passing the same audio clip.

---

void startAudioLoop(AudioClip audioClip, float volume)

Starts the audio clip specified by the parameter audioClip playing on a continuous loop with a volume specified by the parameter volume. The clip can be stopped by calling the function stopAudioLoop and passing the same audio clip.

---

void stopAudioLoop(AudioClip audioClip)

Stops the audio clip specified by the parameter audioClip from playing on continuous loop.

Common Audio Function Usage

The code snippet in Listing 2.3 shows the common usage of loading and playing an audio file.

Listing 2.3: Example use game engine audio functions.

```
. . .

AudioClip  beep ;

AudioClip  background ;

public void  i n i t ( )  {

      . . .

     beep = loadAudio ( " beep . wav" ) ;

     background = loadAudio ( " background . wav" ) ;

     s t a r t A u d i o L o o p ( background ) ;

}


public void mousePressed ( MouseEvent  e v e n t )  {

     i f ( e v e n t . getButton ( ) == MouseEvent .BUTTON1) {
          playAudio ( beep ) ;

     }

}

....
```

2.7 Mathematical Functions

The game engine provides the following functions for performing common Mathematics operations. Trigonom-etry functions are described in the following section.

---

int rand(int max)

Generates a uniform random value and converts it to an integer in the range [0, max). Integers starting from (and including) zero up to (but not including) max will be returned with an equal probability.

---

float rand(float max)

Generates a single-precision uniform random value in the range [0, max).

---

double rand(double max)

Generates a double-precision uniform random value in the range [0, max).

---

int floor(double value)

Converts a double value to an integer by returning the largest integer that is less than or equal to the parameter value.

---

int ceil(double value)

Converts a double value to an integer by returning the smallest integer that is greater than than or equal to the parameter value.

---

int round(double value)

Converts a double value to an integer by rounding the parameter value to the closest integer.

---

int abs(int value)

Calculates the absolute value of the integer int parameter value. Absolute value removes the sign of a value.

---

float abs(float value)

Calculates the absolute value of the single-precision float parameter value. Absolute value removes the sign of a value.

---

double abs(double value)

Calculates the absolute value of the double-precision double parameter value. Absolute value removes the sign of a value.

---

double sqrt(double value)

Calculates the square root of the parameter value.

---

double length(double x, double y)

Calculates the length of the vector (x,y). This is the same as the distance from the point (0,0) to the point (x,y).

---

double distance(double x1, double y1, double x2, double y2)

Calculates the distance from the point (x1,y1) to the point (x2,y2).

2.8 Trigonometry Functions

The game engine provides the following Trigonometry functions. All trig functions take and return values specified in degrees. The methods toDegrees and toRadians can be used to convert between radians and degrees if necessary.

---

double toDegrees(double radians)

Converts the parameter radians from an angle specified in radians to the same angle specified in degrees.

---

double toDegrees(double degrees)

Converts the parameter degrees from an angle specified in degrees to the same angle specified in radians.

---

double cos(double value)

Calculates the cosine of the parameter value.

---

double acos(double value)

Calculates the inverse cosine or arccosine of the parameter value. Parameter must be in the range [-1,1].

---

double sin(double value)

Calculates the sine of the parameter value.

---

double asin(double value)

Calculates the inverse sine or arcsine of the parameter value. Parameter must be in the range [-1,1].

---

double tan(double value)

Calculates the tangent of the parameter value. Result is not defined for values [90 ; 270 ; 450 ...]

---

double atan(double value)

Calculates the inverse tangent or arctangent of the parameter value.

---

double atan2(double x, double y)

Returns the angle theta from the conversion of rectangular coordinates (x, y) to polar coordinates (r, theta).