

Blender Game Engine (BGE)

The BGE is one of the most interesting parts of Blender. It allows the creation of simple games without, if wanted, having to program. BGE uses logical blocks to control the movement and the display objects. These blocks are divided into the following:

- **Sensors** – detect events (e.g., what key or button mouse is used);
- **Controllers** – combine the inputs in order to activate a response;
- **Actuators** – start a task upon receiving a positive response from the controllers.

- Open Blender;
- Change the cube's location to (5, 5, 0.5);
- Change the cube's dimension to (1, 1, 1)
- Change the diffuse colour of the material to blue;
- Change the **Engine** menu to **Blender Game** (Info Window);
- Change the Blender layout to **Game Logic**.

Controlled Navigation

- With the cube selected, press the button **Add Sensor** and select the **Keyboard** sensor;
- Press the **left button mouse** above the empty field in front of the **Key** area and, then, press the key → (this sets the right arrow to this sensor and when pressed it will activated a positive response);
- Change the name of this sensor to "Right Arrow";
- Press the button **Add Controller** and select the **And** controller;
- Change the name of this controller to "And_RA",
- Press the button **Add Actuator** and select the **Motion** actuator;
- Change to 0.1 the value of the X axes, in the **Loc** area (each time that the key → be used, the cube will jump 0.1 unities of Blender);
- Change the name of the actuator to "Mov +X";
- Link the dots between the blocks (from one dot, press the **left button mouse** and, keeping the button down, move the mouse till the other dot);
- In the properties window, icon **Render**, press the **Start** button, in the panel **Embedded Player** (or **P Key**), and verify that when pressing the → Key the cube goes right;
- Return to the Logic Editor, pressing **ESCAPE** key;
- Apply to the cube the following movements, controlled by the keys ← (**Loc X**=-0.1), ↑ (**Rot Z**=2°) and ↓ (**Rot Z**=-2°).

Automatic and realistic navigation

- Change the visualization to camera view;
- Change the camera to the location (13, -15, 10), rotated (60°, 0°, 35°);
- In the icon **Object Data**, **Lens** panel, change the **Focal Length** to 25;
- Add a plane, with green as diffuse colour and 16x16 as dimension, to (0, 0, 0.1);
- Apply to the plane 3 subdivisions and remove the faces as figure1;

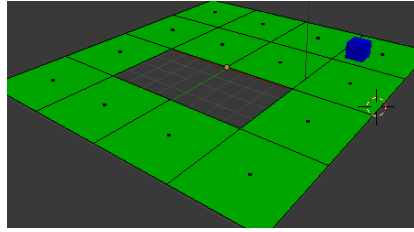


Figure 1: game's 3D plane.


- Add a cone in the position (0, -7, 0.5), with dimension (1, 1, 1);
- With the cone selected, press **Add Sensor** button and choose the **Always**;
- Press the **Add Actuators** button and select the **Steering** actuator;
- Put the name of the cube in the field **Target Object** ;
- Link the blocks and verify (**P** Key) that the cone moves towards the cube, passing over the plane's hole, as it doesn't exist;
- Select the cone and in the icon **Physics** (properties windows), panel **Physics**, change its physics type to **Dynamic**;
- In the **Steering** actuator, deselect the **Lock Z Velocity** button);
- Verify (**P** key) that the cone falls into the plane's hole;
- For the cone go around obstacles like the plane's hole, it should be used some kind of artificial intelligence. To accomplish that, Blender has the technique called **navigation mesh**. Any element to become intelligent should be located within the navigation mesh.
 - Select the plane and in the icon **Scene** (properties window), **Navigation Mesh** panel, press the button **Build Navigation Mesh**;
 - Select the cone and, in the **Steering** actuator, sets the **Behavior** field with the value **Path Following** and the **Navigation Mesh** field with the name of the created navigation mesh;
 - Verify (**P** key) that the cone search the cube without falling in the hole;
 - In Editing Mode, select one edge of the plane and made an extrusion in the XX axes. Apply to the created edge a translation in ZZ axes and made another extrusion in the XX axes;
 - In the Object Mode, put the cube on the top of the created ramp;
 - Verify (**P** key) that the cone doesn't climb the ramp because the navigation mesh doesn't exist there;
 - Select the plane and in the icon **Scene** (properties window), **Navigation Mesh** panel, change the **Max Slope** field to 75°;
 - Press the **Build Navigation Mesh** button and verify that the navigation mesh goes into the ramp;
 - Select the cone and, in the **Steering** actuator, set the **Navigation Mesh** field with the name of the created navigation mesh;
 - Verify (**P** key) that the cone could now climb the ramp, in spite of the movement don't be fully correct (the cone enters inside of the ramp);
 - Select the cone and, in the **Steering** actuator, activate the field **N**;
 - In the icon **Render** (properties window), press the **Start** button in the **Standalone Player** panel and see that the movement's cone was better;
 - Return to the Logic Editor, pressing the **ESCAPE** key.

Playing with collisions

- Select the blue cube, change its physics type to **Dynamic** and verify (**P** key) that the cube changes its actuation (try put it in the middle of the ramp);
- Add another cube, with dimension (1, 1, 1), in the position (-7, 5, 0.5);
- Verify, in the icon **Physics** (properties window), panel **Physics**, that its physics type is **Static**;
- Verify (**P** key) that it's impossible either to change its location using another element nor passing through it (used to create limits to the game);
- Add a red sphere, with dimension (1, 1, 1), in the position (2, 5, 0.5);
- Change its physics type to **No Collision**;
- Verify (**P** key) that the blue cube passes through the sphere;
- Change the sphere physics type to **Dynamic**;
- Verify (**P** key) that the blue cube can't pass through the sphere (just push it);
- With the sphere selected, press the **Add Sensor** button and choose the **Collision** sensor;
- Click in the field **M/P** and in the front area select the blue material. This makes that only the elements that have the chosen material of this sensor, when touched the sphere, triggered a positive response;
- Press the **Add Actuators** button and select the **Edit Object** actuator;
- In this actuator, change it to **End Object** option;
- Link the blocks and verify (**P** key) that the sphere disappear when the blue cube touch it;
- Copy the sphere (**SHIFT + D** keys) and put it in another position, align with the previous sphere in the XX axe;
- Verify (**P** key) that the blue cube made both the spheres disappear, proving that the copy of an element, copy also the associated blocks that the origin had.

Properties

- Properties are some kind of variables that keep information that could be used during the game. BGE has two types of properties, the basic ones (created when the button **Add Game Property** is pressed) and the ones linked to a text element (created when the button **Add Text Game Property** is pressed). This last type allow the users see on the screen the value kept by the property;
- One example for the property linked to a text element could be the following:
 - Create the text element "0" (just the number 0), aligned with the view, with the name "Count" and put it on the right corner of the frame;
 - Press **Add Text Game Property** button (panel **Properties** – Logic Editor);
 - Change the type of the property to **Integer**;
 - Select both, the red sphere and the text element;
 - Add a **Property** actuator to the text element ("Count");
 - Change the field Mode to **Add**;
 - Set the property Text to the **Property** field (the one that was linked to the text element);
 - Put +1 in the field **Value** (adds 1 to the property Count that was initialized with 0, when the text was created);

- Link the **And** controller of the red sphere to the actuator created above;
 - Verify (**P** key) that the counter in the right corner of the frame is incremented when the blue cube touched the red sphere used above;
 - Join the last red sphere to the selected elements and link its **And** controller to the actuator of the text element;
 - Verify (**P** key) that the text in the right corner of the frame has the information of how many red spheres are touched by the blue cube.
- Na example for the use of the basic properties could be the following:
 - Add to the game a plane with dimension 30x30, name “Test” and in the position (0, 0, -2);
 - Press the **Add Game Property** button (panel **Properties** – Logic Editor);
 - Change the name of the property to “life” and its type to **Integer**;
 - Change the value of the property to 1;
 - Press the button  (allows to see the value of the property in the debug mode);
 - In the **Info** window, select the **Game** option and activate the button **Show Debug Properties**;
 - Add a **Collision** sensor and a **Property** actuator to the “Test” plane;
 - Change the actuator field Mode to **Add**;
 - Set the field **Property** with the property “life”;
 - Put -1 in the field **Value** (subtract 1 to the value of the life property);
 - Link the blocks and verify (**P** key) that when the blue cube touch the “Test” plane the property “life” pass from 1 to 0;
 - With the “Test” plane selected, add a **Property** sensor;
 - Put in the field **Property** the property “life”;
 - Set to 0 the field **Value** (will activate a positive response when the property “life” was equal to 0);
 - Add a **Game** actuator and change the **Game** field to **Quit Game**;
 - Link the blocks and verify (**P** key) that when the value of the property “life” become 0 the game ends.
- The plane “Test” only serves to optimize the game and must be invisible. thus:
 - Select the plane;
 - In the icon **Physics** (properties window), panel **Physics**, press the **Invisible** button;
 - Verify (**P** key) that the plane is not rendered.

Using different cameras in the game (First camera Shooter)

- Usually the games has several cameras to show the scenes from different points of view. One example of how to do this is the following:
 - Add a camera, named “CamGo”, in the same position of the blue cube and rotated (90°, 0°, 90°);
 - Select the camera and, then, the blue cube;
 - Create a parent relation by pressing the keys **CTRL + P** and choose the option **Object** (when the cube moves the camera moves accordingly – to create this type of relations always must be selected first the sons and only then the father);

- Select only the camera and add a **Mouse** sensor;
- Change the field Mouse Event to **Right Button**;
- Add a **Scene** actuator and change the field Mode to **Set Camera**;
- Put in the field **Camera Object** the name of the created camera;
- Link the blocks and verify (**P** key) that when the right button of the mouse was pressed the view of the game changes;
- Do the same procedure to return to the previous camera when press the left button mouse.

Screen Menu

- The games usually start with a main menu and then go to the game scene, when pressed a button or a key. It is called Screen Menu. A menu of this kind can be made as follows:

- Create a new scene, called “MenuIntro” (button + in the Scene area – Info window, followed by the option **New**);
- Change the **Engine** to **Blender Game**;
- Add a new camera, located in (10, -10, 7) and rotated (60°, 0°, 35°);
- Change the visualization to camera view;
- Add a plane, align to view, in the position (2, 1, 0);
- Add a **Mouse** sensor to the created plane;
- Change the Mouse Event field to **Mouse Over** (activate a positive response when the mouse cursor passes over the plane);
- Add a **Scene** actuator, set the Mode field with **Set Scene** and in the Scene field put the name of the other scene;
- Link the blocks and verify (**P** key) that the mouse cursor doesn’t appear;
- In the icon **Render**, properties window, panel **Display**, activate the option **Mouse Cursor**;
- Verify (**P** key) that now the mouse cursor appears and when passes over the plane the scene is changed;
- This way is not common in a game, so add a **Mouse** sensor to the plane;
- Link the previous sensor to the controller **AND** that is linked to the other **Mouse** sensor;
- Verify (**P** key) that the scene only changes when the mouse cursor is over the plane and the left button mouse is pressed.

Actuator Action

- It can be use in games any type of animation (made with keyframes, ShapeKeys or based on armatures). An example of this can be done as follows:

- Change the layout to **Animation** and select the scene MenuIntro;
- Change the visualization to Camera View and Select the plane;
- Set the frame of the **Timeline** window to 1 and create a keyframe (**I** key – option **Scaling**, if in the 3D View window) with the plane as it is;
- Set the frame to 24, scale the plane 5 unities in the XX axe (area transform in the panel properties – **N** key) and mark a keyframe;
- Set the frame to 48, scale the plane 1 unity in the XX axe (area transform in the panel properties – **N** key) and mark a keyframe;

- Change the layout to **Game Logic**;
- With the plane selected, add an **Action** actuator;
- Put in the area below the **Play** the name of the created action and set the field **Start Frame** to 1 and **End Frame** to 48;
- Link the action actuator to the **Mouse Over** sensor;
- Verify the result when the mouse cursor passes over the plane (**P** key).

Creating the executable

- In order to create the game's executable, it must be active the export project to executable addon. If it is not active, it must be the following:

- In the Info window, select the option **File → User Preferences**;
- Select the panel **Addons**;
- Press the **Game Engine** button;
- Activate the addon **Game Engine: Save As Game Engine Runtime**;
- Close the User Preferences window.

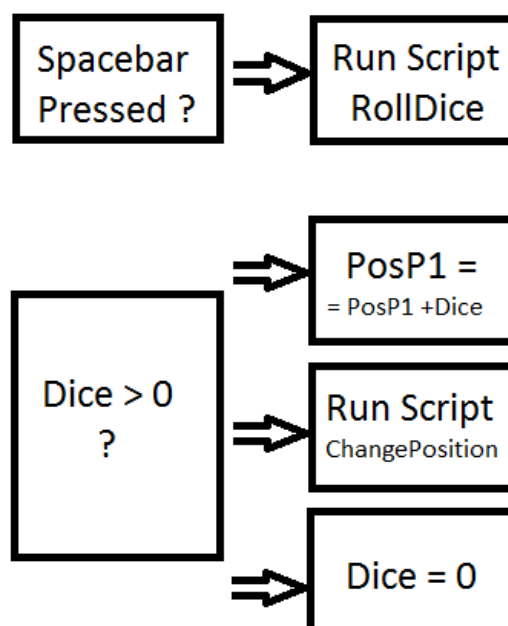
- Having activated the export project to executable addon, the creation of the game's executable is made as follows:


- In the Info window, select the option **File → Export → Save as Game Engine Runtime**;
- Select the directory where you want to put the game's files and the name of the executable;
- Press the **Save as Game Engine Runtime** button;
- Verify if the directory was created and play the game.

Using de scripts

- In most cases, when creating a game, specific operations must be programmed. In BGE this is done by **scripts** that use the Python language. An example of its use can be done as follows:

How to move the pawn on the board?




- Open the file “FCG_05_BGE_Tabuleiro.blend”;
- To obtain a smooth movement for the pawn, add an **empty** element (**add → Empty → Plain Axes**) to the scene in the position (0.0, 10.0, 0.5);
- Select the game board and in the icon **Scene** (properties window), panel **Navigation Mesh**, press the button **Build Navigation Mesh**. Thus the pawn never leave the board;
- With the pawn selected, add an **Always** sensor;
- Press the **Add Actuators** button and select the **Steering** actuator;
- In the **Behavior** field choose **Path Following**, in the **TargetObject** field put the name of the empty element, in the **Navigation Mesh** field put the name of the navigation mesh created, set the **Dist** field with value 0 (goes to the exact position of the Empty element) and activate the button **Visualize**;
- Link the sensor to the actuator;
- Select the Empty element and in the panel **Properties**, of the Logic Editor, press the **Add Game Property** button;
- Change the name of the property to “Dice”, the type of the property to **Integer** and press the button  (to allow the visualization of property's values in the debug mode);
- In the **Info** window, choose the option **Game** and activate the button **Show Debug Properties**;

- A widely used function in the games is the assignment of a random value to a variable (for example, to simulate a dice). As the BGE has no predefined blocks to do this, it must be used a script (a little program). Thus,

- In the **Text Editor** window, press the **New** button and write the following:


```
import random          (to be able to use the random library)
from bge import logic   (to be able to use the script with BGE blocks)
cont=logic.getCurrentController() (to obtain game's information)
own=cont.owner          (to obtain object's information to use in the script)
rand=random.randint(1, 2) (Generate random value between 1 e 2)
own['Dice']=rand        (set the property Dice with the random value)
```
- Change the **script** name to “RollDice”;
- With the Empty selected, add a **Keyboard** sensor;
- Press the **left mouse button** on the empty field in front of the area **Key** and, then, press the **SPACEBAR** key;
- Enable sending only a positive response to the controller, even if the key be hold down continuously, by pressing the **Tap** area;
- Add a **Python** controller and in the field in front of the word **Script** put “RollDice”;
- Link the blocks;
- Verify (**P** key) that when the **SPACEBAR** was pressed the property “Dice” change its value to 1 or 2.

- One of the ways to put the pawn to walk on the squares of the board is made as follows:

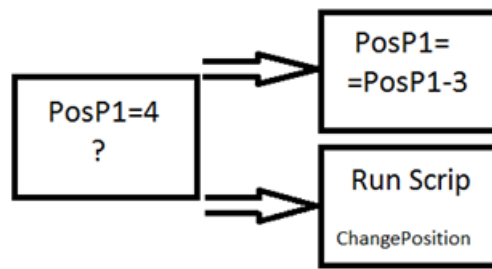
- Select the Empty element;
- In the panel **Properties** of the Logic Editor, press the **Add Game Property** button;
- Change the property's name to "PosP1";
- Change the property's type to **Integer** and press the button .
- In the **Text Editor** window, press the + button and write the following:

```
from bge import logic    (to be able to use the script with BGE blocks)
cont=logic.getCurrentController()    (to obtain game's information)
own=cont.owner    (to obtain object's information to use in the script)
if own['PosP1']<=6:    (Define the initial pawn movement)
    own.position = [0.0, -own['PosP1']*2.0+10.0, 0.5]
elif own['PosP1']>=7:    (Define the final pawn movement)
    own.position = [-(own['PosP1']-6)*2.0, -2.0, 0.5]
```

- Change the **script** name to "ChangePosition";
 - With the Empty selected, add a **Property** sensor;
 - Put **Greater Than** in the field **Evaluation Type**;
 - Put "Dice" in the **Property** field;
 - Put 0 in the **Value** field (a positive response is triggered when the property "Dice" is 0);
 - Add a **Python** controller and in the field in front of the word **Script** put "ChangePosition";
 - Link the blocks;
 - Add a **And** controller and link it to the sensor Property (Dice>0);
 - Add a **Property** actuator;
 - Change the field **Mode** to **Add**;
 - Put in the field **Property** the property "PosP1";
 - Put the property "Dice" in the field **Value** (sum the value of the property "Dice" to the property "PosP1");
 - Link the created actuator to the previous **And** controller;
 - Add a new actuator **Property**;
 - Put **Assign** to the field **Mode**;
 - Put "Dice" in the **Property** field;
 - Put 0 in the field **Value** (sets that value to the property "Dice");
 - Link the created actuator to the **And** controller where the previous actuator was linked;
 - Verify (**P** key) that when the **SPACEBAR** is pressed the pawn moves accordingly on the board;
- It is important to note that the formulas used to move the pawn depends on the position of the squares of the board;
 - It is important to note that the order of the blocks is essential. In this case, if the block that sets the property "Dice" to 0 was first then the one that adds the "Dice" property to the "PosP1" property, the pawn never move.
- During a board game, tend to be places where one has to retreat, advance, answer a question, got to the start or finish position or take any other action. So, the treatment, for example, the situation of going

back when the pawn arrive to the yellow square on the board, can be done as the following:

How to go back 3 squares?



- With the Empty selected, add a **Property** sensor;
- Put in the **Property** field the property "PosP1";
- Set the **Value** field with 4;
- Link the created sensor to the "ChangePosition" Python controller;
- Add a **Property** actuator;
- Change the field **Mode** to **Add**;
- Put in the field **Property** the property "PosP1";
- Put -3 in the field **Value** (go back 3 squares);
- Link the last created sensor to the actuator;
- Verify (**P** key) that the pawn go back three squares when arrive to the yellow square.