A Survey of Game Editors

Zachary Shutters and Paul Gestwicki
Computer Science Department
Ball State University
Technical Report 2007-001

Game creation is no longer in the realms of the elite programmer. Anyone with enough motivation to make a game will find there are many tools available that allow them to create complex games with little to no programming experience. Of all the available game engines, I will discuss the ones that are geared toward the non-programmer. I will evaluate them based on their ease of use, available documentation, online support, and how easily a game can be made from the perspective of a non-programmer.

Adventure Game Studio

With Adventure Game Studio (AGS), you can create point and click adventure game, akin to LucasArts game Monkey Island. The game supports resolutions of 320x200, 320x240, 640x400, 40x480, and 800x600, with bit depths of 256, 16, and 32. Sound formats supported are OGG, MP3, AV, MOD, XM, and MIDI. The game engine itself has 3 different versions – Windows, Linux, and MS-DOS, but you can only create the game within Windows.

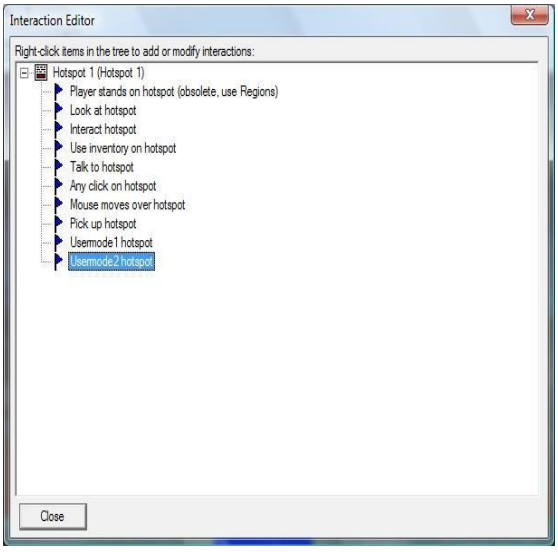
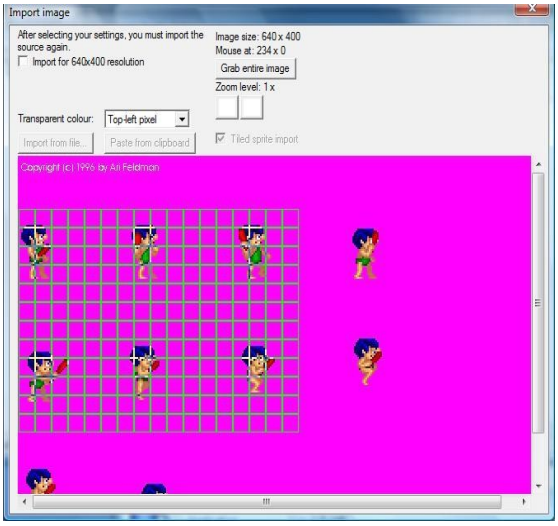


Figure 1

Creating a game with this game engine revolves around creating rooms and filling them with objects. You have precise control over where the player can walk in a room by setting walkable areas. Then, you can fine tune the room more by adding hot-spots. These are areas in the room that trigger events to happen such as making a light in a room turn on, or trigger a monster to come out and attack you. If you look at fig.1, you can see a menu that allows you to choose how the hotspot will be triggered. Once you choose how the hotspot is triggered, you then get to associate an action with the hotspot, which is done by another menu that pops up with a long list of actions that can be performed.

Objects are things that you can put into the rooms that move on their own accord or have the ability to be manipulated by the player. To put an object into your room, you first must create it by clicking on objects, under room editor, and then click new object. From here, you place the image anywhere in the room, select the image used to represent the object, and then set up interactions with the object. Setting up interactions with an object is similar to setting them up for hotspots; you choose from a list of predefined interactions such as, look at, pick up, or talk to. You are not limited to just one interaction per object; you can have multiple interactions. The game engine automatically controls an inventory list for you so when you pick up objects, you can set an action to add them to your inventory. AGS comes with a few graphics to get you started, but you must make or download your own if you want to make a decent game.

Figure 2



Within the game editor, you can import images to use as sprites. When you import an image, you have an option of importing the entire image or portions of it by dragging a grid over the image, as shown in fig. 2. The editor will then import each grid square as a separate image. This feature would be useful with image strips, but unfortunately, it is unclear on how to change the grid size.

You also have the ability to do text dialogues within the game, to show your character talking to another character. AGS handles conversations by treating them as topics – a set of items the player can talk about. After creating a new topic, you can create a list of questions to ask another character, but then to create the response the character gives, you must edit a script file. An example of this file is given below, beside fig. 3. If you look at the option names in fig. 3, you can see that they correspond to the responses in the script file.

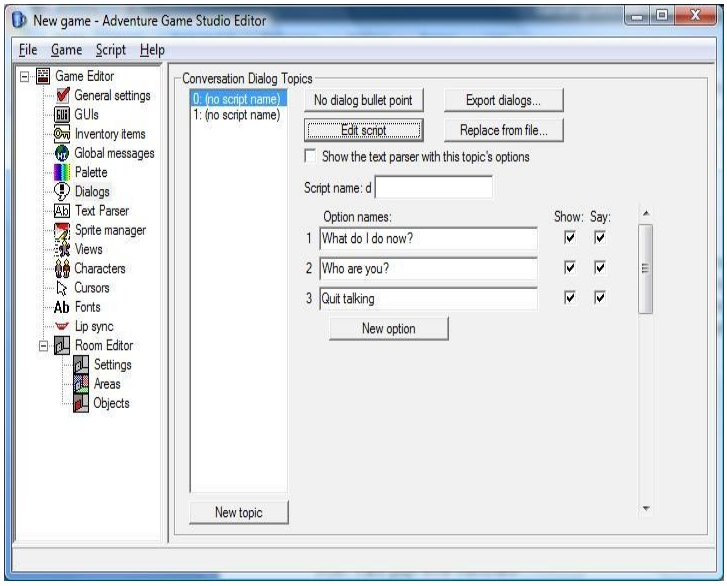


Figure 3

```
// dialog script file
@S  // dialog startup entry
point
EGO: "Greetings!"
JOE: "Hello there!"
return
@1  // option 1
JOE: "Go find the yellow key"
return
@2  // option 2
JOE: "I am your local
merchant"
return
@3  // option 3
JOE: "Goodbye!"
```

This pretty much sums up the development process with AGS. There is a scripting language that you can use to make more complex games, and it is similar in style to C programming language. An entire game could be made with just the scripting language. Though for the non-programmer, everything can be done with a point-and-click fashion, with a few exceptions – for instance, the dialogs.

Documentation of this game engine seems to be complete and thorough, and there is also a very active forum on their web page. There is a little bit of a learning curve in creating games with this engine, but overall, the difficulty level is not very high. This engine allows you to create an adventure style game, concentrating solely on game play and graphics.

Silent Walk FPS creator

Silent Walk FPS creator (SWF) runs on Microsoft Windows running DirectX 8.0 or above. A 3D graphics card with at least 16 MB of video memory is also required. You can download an unregistered version of SWF with full functionality, but when you play your game, a translucent splash screen appears as an incentive to register your copy. Registration fees are $15 for the current version (version 1.1) and $19.99 for the current version and a pre-order for the next release (version 2.0). There is also the option to buy the source for $999. There is not much information on the SWF website about this game engine and mainly relies on the forums for users to get information. The forums seem to be very active, and there is a lot of good feedback to questions asked. From information that was gathered from the forums, it appears that SWF was created from another game engine I have reviewed, GameMaker; taking advantage of its extensibility with being able to load custom dll's to add new functionality to it.

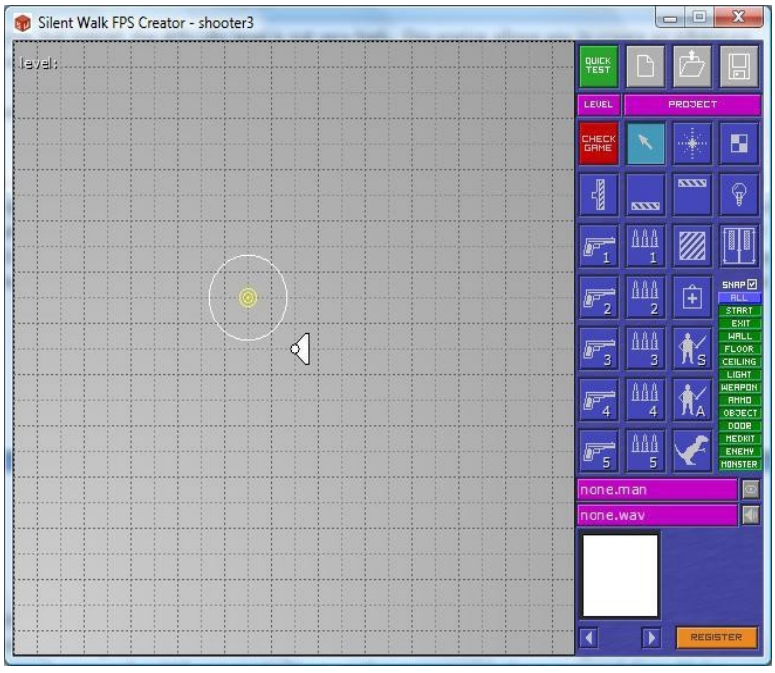


Figure 4

SWF is an engine for making first person shooter games. It only supports a custom model format, but they do offer an additional download (free) that will convert WaveFront .obj, 3D studio . 3ds, and DirectX .x models to their proprietary format .man. On their website, they offer four resource packs to download for free to help get you started with your game – sound effects, music, textures, and models. There are no restrictions on these resources other than they can only be used for SWF games.

Designing a game with SWF is really simple, but there are a few quirks to get started. There is no installer for it –just a zip file with all files. First, you must extract the files to the destination of where you want the program to reside. Among the game engine files is a folder called projects, and this is where your game will saved. Unfortunately, you cannot change this to a different folder. When you finish creating a game or want to test it out, you must first close the program. Then navigate to this folder to execute your game, which is another annoying problem. Within the editor, you can launch your game, but it is only for previewing; no game play functionality is enabled other than walking around.

The game engine takes care of a lot of things automatically in order to shield the developer from having to write any code at all. There is a customizable title and exit screen, as well as the ability to play an avi or mpg movie at the beginning and end of the game. Mapping of the controls to the keyboard is done automatically, though it is customizable through a menu within the editor. Keeping track of ammo and health and displaying these in the HUD is also taken care of by the game engine. All the sounds have default settings that are customizable too. Since only FPS game can be made with this engine, there really isn't much else that needs to be done. You can make a game where you run around and shoot people and pick up keys to unlock doors. That's pretty much the mantra for FPS games.

Creating your levels and adding objects within them, is a simple point and click process. You have seven objects from which you can use to create your levels – start point, exit point, light, wall, door, floor, and ceiling. When you start building your level, you are presented with a grid to build on, which is a top down representation of your 3D level. This is not the best perspective, but since currently only one layer levels are supported, it is not too troublesome. Version 2.0 supposedly will support multiplayer levels, so it will be interesting to see if the perspective for development is changed. To add an object to the grid, you just click on which one (object) you want and click anywhere on the grid that you want it placed. This process is a little tedious when you are trying to make a large level because you have to click on each grid square to place each piece of the floor. Then, go back over the same grid squares to place each piece of the ceiling. You use this same process for placing the rest of the types of objects.

There are three categories of enemies that you can place – an enemy that stays in the same place, one that will chase you and then try to hide if its health gets too low, and finally the kind that will chase you nonstop. The game engine takes care of the AI routines; that part is completely hidden from the developer.

You can use up to five different types of weapons in your levels, and SWF refers to then as weapon1, weapon2, weapon3, weapon4, and weapon5. These weapons are automatically mapped to the keyboard keys one through five; the mouse scroll wheel is also tied to switching weapons. The weapon models can be of any model you design or download. There is also an ammo model, that you must design or download, that is tied to each weapon. For example, an ammo model for weapon 3 would only supply ammo

for that weapon, not any of the others. The weapon choices do lack the ability to zoom in if there is a scope on the weapon, and currently there is no way around for this.

Documentation for this game engine is short and to the point. On their website there is a manual for the editor and a tutorial for making a game. The manual does not quite cover how to use everything in the editor, but the tutorial for making a game does supplement this well. Although there is still room left for confusion, a few hours of playing with it clears most of this up.

Overall, this turned out to be an impressive engine on the basis of absolutely no programming knowledge being needed. Anyone who has a little motivation and creativity can easily use this engine to make a FPS game.

GameMaker

GameMaker comes in two different versions, pro and lite. The lite version can be downloaded for free, but a lot of functionality is disabled; you can still make a decent game with the lite version, but you end up doing a lot of tricks to get around the limitations. GameMaker can only be run on Microsoft's Windows operating system -- Windows me and above running DirectX 8.0 or above. Hardware requirements are a 32mb video card and a DirectX compatible sound card. An attempt was made to run GameMaker on Ubuntu Linux under Wine and CrossOver Office with no success.

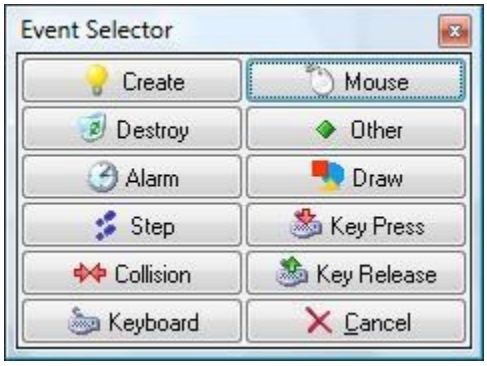
GameMaker's game engine consists of a 2D engine with features such as color blending, translucency, and particle systems. There is also a 3D and multiplayer engine, but they both require using GameMaker's proprietary scripting language GML. GameMaker's strong point seems to be with 2D games whereas Torque and 3D GameMaker are more suited towards 3D games.

GameMaker provides a development environment that is intuitive and easy to use. People with no programming experience should not have any problems learning their way around the program after following through a few 10 minute tutorials. I went through one tutorial on how to make a simple pong type game and was then able to make a basic Pac-Man game within 4 hours. Though, now that I am more experienced with using the program I could make another simple game within half an hour, assuming all the graphics I needed were already available.

In GameMaker, everything in your game that moves, besides scrolling backgrounds, are treated as objects, which is one of the seven basic resources available. The other eight are sprites, backgrounds, rooms, paths, scripts, fonts, time lines, and sounds.

Sprites are used to represent images that can then be used to represent objects. These images can be either image strips or a single image. If using image strips, then GameMaker automatically cycles between the different images to create the animation effect. When creating a sprite, you can either use images you've made with another program or create the image with the built-in drawing program. It is not a very sophisticated program, but it does allow you to create images with a transparent background and easily create image strips. If you choose to create your own artwork, GameMaker supports over 40 different image file formats, which includes just about every format used with most Windows drawing programs.

Figure 5



Objects are the most important part of GameMaker. To fully understand objects, I have to first talk about actions and events. You can think of objects as being event driven; you associate events with the object. Then, when that event happens, the object performs a set of actions that you have specified. By doing it this way, GameMaker has succeeded in developing a way to add interactivity to your game without the need to learn any programming. Objects can even be the parent of other objects. For example, if you wanted to create five different enemy ships that all do the same thing but have different images associated with them. You could create an object that represents all enemy ships and register all the events and actions that all enemy ships should do (notice that I do not associate a sprite with this object since it only serves as a template for enemy ships). Next, you would create each enemy ship and set them to be the children of the template enemy ship object. In doing this, each of the children ships inherit all of the events and actions of the parent but can have their own unique sprite image. You can also override the inherited events by defining new ones within the child object.

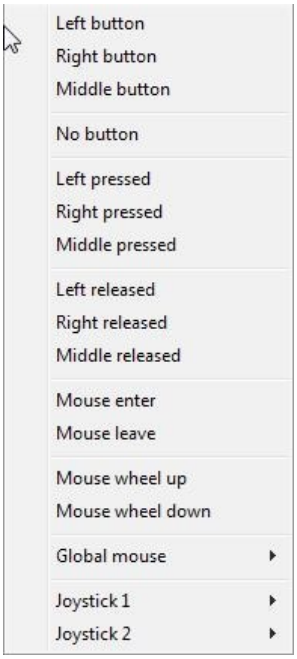
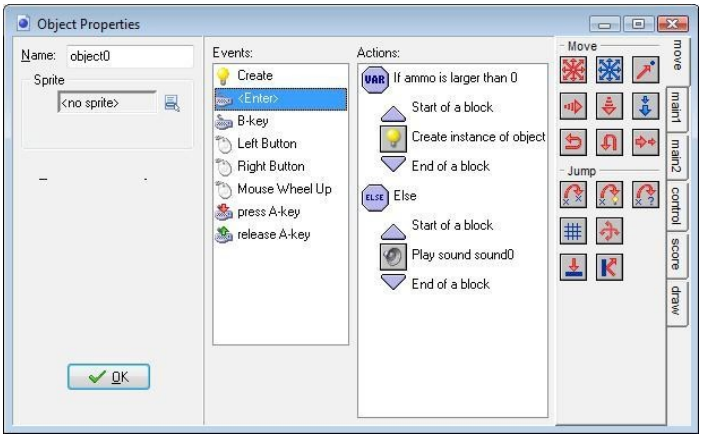


Figure 6

There are eleven basic events to choose from when registering events with your object. From these eleven, seven of them have submenus to pick more detailed events. For instance, the mouse event has a submenu, as shown in fig. 6, which lets you choose from many different kinds of mouse events. The eleven basic events are shown above in fig. 5. With these events, you are able to react to most things that could happen to an object within a game. If, for some reason, you come across an event that is not fit into one of these categories, there is the option of creating your own user defined events, but requires learning GML, GameMaker's proprietary scripting language. The name of the events describes what type of event on which they are reacting. A few of them are not so obvious at first. The destroy event executes its actions when the instance of the object isdestroyed. The alarm event executes its actions when an alarm, which has been created in the action of another event, reaches zero. You can have up to twelve different

alarms, which can be used for things like creating a delay, or randomly generating objects. The step event provides a way for your game to execute actions with each loop of the game. The events are similar to the way 3D GameMaker allows you to add behaviors to 3D models.

Figure 7



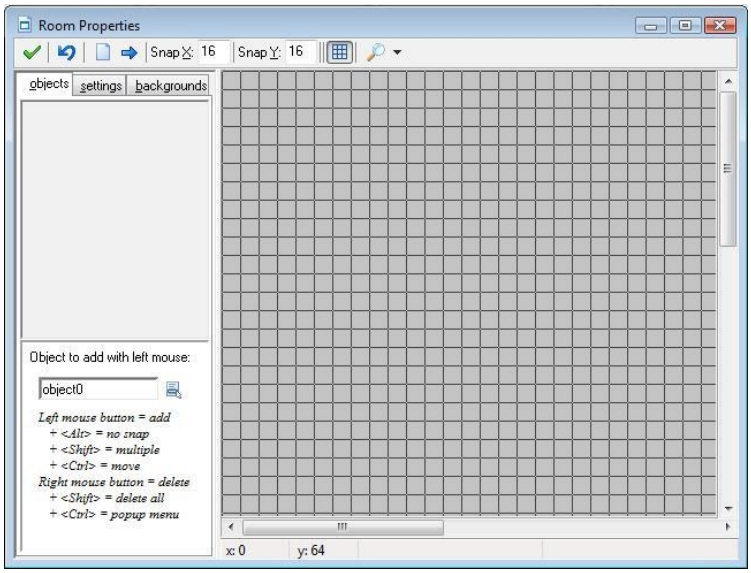
Once you have an event selected that you want your object to react to, you then add actions to the event. Looking at figure 7, you can see there is an area for events and actions. Along the right side of the figure, there is a tabbed menu to choose different types of actions to perform. To add an action to an event, you must first add the event to the object. Then, drag and drop actions from the tabbed menu to the actions area. When you do this, a window will pop up with settings based on the action you are adding. You can also do if else statements within the actions menu. For example, on a keypress event, you want the player character to shoot his gun (if there is ammo left). On the control tab, there is an icon for test variable. When you drag this to actions for the keypress event, a window will pop up that will allow you to type the variable name, the value you want to check it against, and whether it should be greater than, equal to, or less than that value. Just like you would expect, if it evaluates to be true, the very next action will be performed, otherwise it will be skipped. There is also an icon for the equivalent of the curly braces found in most programming languages to denote a block of code. In GameMaker, these are represented by a triangle pointing up for the opening brace and a triangle pointing down for the closing brace. Using these after testing a variable will cause all of the actions contained between them to be executed if the expression is evaluated as being true. As you can see in figure 7, there is also the else clause which will execute its block of code if the if statement is evaluated as false. To declare variables, you drag and drop the variable icon to the creation event. Since objects can be thought of like classes in c++ and java, the variables can be thought of like instance variables. You can then make them accessible from other objects by prefixing global to the beginning of your variable name, separated by a dot. Therefore, without the global prefix, the variables are treated as private.

Every object automatically has certain variables declared for it and are kept updated by the game engine. Some of the more useful of these variables are x, y, hspeed, vspeed, speed, and direction. Where x and y represent the Cartesian coordinates of the object, hspeed and vspeed represent the vector components of the speed variable, and direction has a value representing the direction of movement of the object in degrees (0-360). There is also a set of global variables that is maintained by the game engine. A complete list of these and instance variables can be found in reference manual.

Some of the global variables keep track of things like the score, lives, mouse position, room width and height, and room caption. Although variables are necessary to

keep track of data in order to create more complex game play, it may be confusing for the non-programmer to grasp. However, there are plenty of tutorials and a very active forum on GameMaker's website for additional help.

Figure 8



Rooms, another of the resources available are also very important in GameMaker because they are where are the objects are placed. Creating a room is straight forward; click the resource menu and click create room. You are then presented with the window depicted in figure 8. It is on this canvas that you build a level for your game. Under the objects tab you will find the objects that you have created. To put them into your level you just select the object then click anywhere on the room and it will place an instance of your object in the position in the room. Under the settings tab you can set the size of the room. Under backgrounds you can use a background resource to set a backdrop for your room. You also have the option of using a background as a tile set and building your background from the tile set. One limitation to this is that you cannot have collision events with tile sets, so you must use other methods if you want a collision with a tile. A nice feature with the backgrounds is the ability to assign a vertical and horizontal speed to it so that is appears the background is moving. This is achieved by wrapping what goes off the screen back to the beginning.

The final resource to discuss is scripts. This resource is GameMaker's proprietary scripting language (GML) which is similar to c. You can make complex games in GameMaker without the need to write any scripts at all. As long as you can think in logical steps, most things can be accomplished with the event/actions process.

Overall, making games with GameMaker is very straightforward and easily lets the beginner jump right in and make what would otherwise take hours of coding. Some of the core concepts of GameMaker introduce you to objected oriented design. Once feature that would be nice to see in future versions would be the ability to save the objects you have created into library. So then you could re-use the objects you have created for use in other games.

3d GameStudio

3D GameStudio is made by Conitec, a German company founded in 1985. GameStudio's game engine uses directx 9.0c or above, and is only available for Microsoft's Windows operating system. The versions of Windows that GameStudio will run on are Windows me, 2000, XP, and Vista. The minimum hardware requirements are a Pentium 3 running at 500 MHz and a 3D accelerated video card with at least 32mb ram.

GameStudio's game engines consists of a 3D engine, particle and effect engine, physics and collision engine, 2d engine, sound engine, and network engine. The physics, sound, effect, and network engines are either limited in functionality in the non professional versions or not included. The game engine also includes support for their proprietary scripting language Lite-C.

To use Lite-C with GameStudio, you have to download an additional package. This package includes an editor, compiler, and debugger. The editor appears to be well developed and contains things like syntax highlighting and built in debugger. Lite-C is similar to c/c++ but claims to be easier to learn, with things like memory and pointer handling managed by Lite-C. On GameStudio's download page is a tutorial in pdf format to get you started in using the language. An example of a simple Lite-C program taken from this tutorial is shown below.

```
////////////////////////////////////////////////////////////////////////
// small.c - small lite-C example
////////////////////////////////////////////////////////////////////////
#include <acknex.h> // include these predefined, needed files in our project
#include <default.c>
////////////////////////////////////////////////////////////////////////
void main()
{
   // Load the level named "small.hmp"
   level_load("small.hmp");
   // Wait for 3 frames, until the level is loaded
   wait (3);
   // Now create the "earth.mdl" model at x = 10, y = 20, z = 30 in our 3D world




   ent_create("earth.mdl", vector(10, 20, 30), NULL);
   // NULL tells the engine that the model doesn't have to do anything
}
////////////////////////////////////////////////////////////////////////
```

GameStudio associates a script (Lite-C code) with everything that moves within your level. A set of basic template scripts can be used by people who do not want to take on programming.

GameStudio provides three programs for game development -- world editor (WED), model editor (MED), and the script editor (SED). WED is where you create your world and add objects to it, like lighting, sounds, models, prefabs, etc.

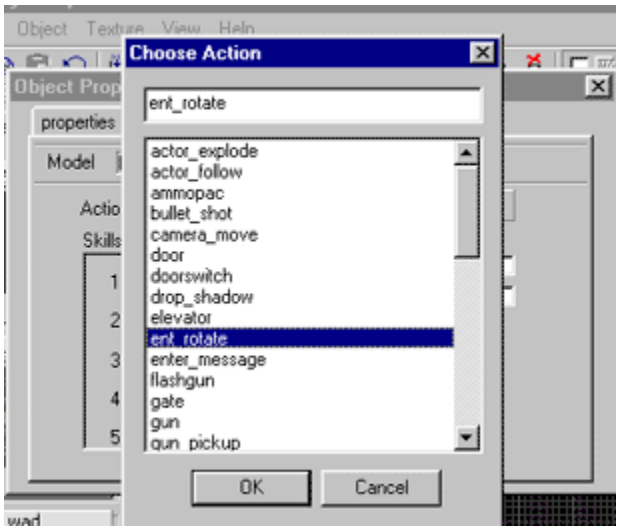


Figure 9

WED provides a completely drag and drop type environment for creating levels without the need to write any code, though you have the option of writing script code (Lite-c) to enhance your game. GameStudio gets away with this by having built in script templates you can choose from. For instance, one such template is a shooter template. What this does is uses a set of prewritten script that contains code to handle things like key presses, score handling, in game menus, and give behaviors to map entities. Behaviors are actions that are more prewritten scripts that allow you to add interactivity to your game and between game objects. There are too many behaviors to list, but they should be able to take care of just about anything you want to do. In fig. 9, there is a window that contains a subset of some available actions that can be added to an entity. Within WED, you get four different views -- top, back, side, and 3D. The 3D view can be switched between wireframe, solid, and textured, whereas the other three views are wireframe only. All four views can be expanded to take the whole screen or shrunken to allow more room for the other views. Objects in the game are represented by primitives, prefabs, light, sound, cameras, terrain, 3D models, 2D sprites, and actor paths. Primitives are the most basic item you can use, but rather than try and build all your own 3d objects from primitives, GameStudio provides a huge library of prebuilt prefabs, which are just map files that have been saved as 3D objects to be inserted into other maps. For lighting, dynamic and static lighting is supported. Static lighting would be a light source that cannot move, but gets rendered realistically and also is able to throw shadows. Dynamic lighting would be any kind of lighting that needs to move, like a torch or candle flame. They are not rendered as realistic as static lights and cannot throw shadows. For 3D models, you have to use MDL format, which is the same format as Quake models. You can also easily download models and prefabs from the Internet and use them in your game creation.

MED, the model editor, as its name sounds, is a tool that lets you create your own 3D models with vertex animation or bone animation. Bone animation is more suited for humanoid type movements, but its file size is larger and consumes about 50% more memory and processing power
while rendering, compared to the vertex animation. There is also a built in skin editor used to apply images to the models you create.

SED, the script editor, is comparable to the average IDE environment. It has standard items like syntax highlighting and a built in debugger. On one side, it has a tree like explorer window that allows you to have an overview of the current script you are working on. There are plenty of tutorials available on GameStudio's website that teaches how to program with Lite-c.

Making simple 3D games with GameStudio appears, at first, to be really simple. Creating levels is about as simple as creating something with logos. But once you start to try and add prewritten scripts, things can get a little confusing. GameStudio's beginner tutorial found at www.conitec.net/tutorial/ tries to walk you through adding a model to your level and then adding behaviors to it. But in following their steps, the model they use does not exist. So in order to try and complete the tutorial, I downloaded a model to use off the Internet but then was not able to add the behavior to it that the tutorial specified. It did not even exist in the list of actions from which to choose. I think the problem is that the tutorial was written for an earlier version of GameStudio, though I could not verify this since the tutorial never mentioned which version it was written for other than the engine version it was using, which was an outdated version. Next, I started looking into how to make a 2D game with GameStudio. I could not find any official tutorials on how to do this, even GameStudio claims to have a 2D Engine; even the built in help system does not mention 2D games anywhere. Due to the scripting language being outside the scope of this paper, I did not attempt to create a 2D game.

Conclusion

All four of these game engines serve the purpose of allowing the non-programmer to create games without the need to learn a programming language, though some do a better job at it. 3DGameStudio has the most advanced tool package for the creation of levels, models, and scripts, but it lacks in ease of use and the learning curve is a bit higher. Using the online tutorials and official tutorials, can be a bit confusing but perhaps a book dedicated to this engine would clarify things a bit more. Whereas, with GameMaker and Silent Walk FPS creator, after working through a few tutorials to learn the basics of using the program, creating a game is very intuitive. Though GameMaker does not completely get away from having to do some scripting, it does simplify the process tremendously by allowing the scripting to be done visually through the use of drag and drop conditional statements. However, since you must also create and use variables, an introduction to variables and how they are used is still needed in order to script effectively. In comparison to GameMaker, the development environment of Silent Walk is a little less intuitive due to the GUI being a bit confusing, and documentation lacking. Though, with a little experimentation, most things can easily be figured out.

3DGameStudio and Silent Walk FPS creator both get away from having to do scripting by having premade scripts that control things that would be needed, like how much ammo is left, or how much health and armor is left. SWF does this naturally since this engine revolves around being a first person shooter engine. Even though there is the option to write you own custom scripts with 3DGameStudioit is not mandatory. There are templates that you can choose from that have prewritten scripts to control things that are common to those types of games. If you look at fig.10 you can see the available types of templates to choose from.

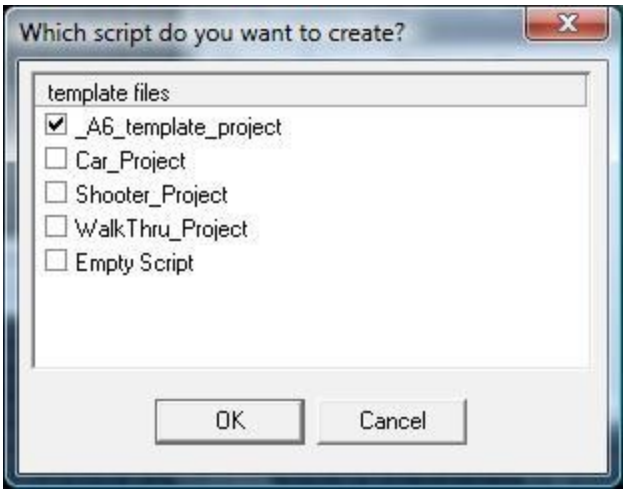


Figure 10

Adventure Game Studio is similar to SWF only in that it is used for a specific genre of games, adventure games. It is able to automatically keep track of and control all the data you would want to with an adventure type of game. Its' interactions and actions that can be associated with objects are very similar to GameStudio's events and actions; they both allow logic to be scripted into the game without the need to write a line of code. AGS though, is not as intuitive as GameStudio, and some time spent reading the manual and going through a few tutorials is needed in order to become proficient with it. It is also not as easy to import and create graphics for sprites as it is in GameStudio and the documentation in this area is somewhat lacking. In looking on the online forums, this apparently causes trouble for a lot of newcomers to the engine. Though, for the most part AGS has solid documentation and tutorials available.

Overall the game engine that impressed me the most was GameStudio. It has a very intuitive interface, which just makes sense all around. The ability to create parent objects, that have children which inherit all their events and actions is a really neat feature, and helps to teach object oriented programming. It has the ability to expand its functionality by writing custom dll's, and it has network programming, though you must learn the scripting language to take advantage of this. A few downsides to this engine are networking, making 3D games and the dependencies on DirectX. Since it is dependent on DirectX we will probably not see a port of it to Linux anytime soon. Making 3D games and network programming is not as intuitive of a process and requires learning how to use the scripting language.