How To Write Good Quality QBasic Action Games

by Jack Thomson
PDF Conversion by Thomas Antoni – www.qbasic.de

Table of Contents

PREFACE

QBasic is a great language if you are learning how to program. It can do a very wide variety of things, including virtually everything you need to program a good quality computer game. Now there are some limitations to QBasic because it's what is called a high level language. It is more English than machine code (bunches of 1's and 0's) by far. Since the interpreter has to convert the "almost English" code into what it can understand, then into machine code, it does take a noticeable amount of time for a certain number of lines of code to complete running. Still, I have written many games in QBasic, proving that it is good for something. This tutorial can be used as sort of a guide

to writing computer games. This article focuses on what is involved in writing action games action games. (NOTE: The concepts here are not limited to QBasic, but apply for any language in which games can be written.)

This tutorial is written for the purpose of explaining to a programmer some of the concepts of planning, creating and programming an action computer game.

ACTION GAMES: OVERVIEW

Assuming that the users processor is fast enough, programming action games in QBasic can be a great thrill. On a 386, some games can run a tad or more slow, but anything from a 486 and higher can pretty much cover any QBasic game that has reasonable enough programming. There are some things that are common to almost all action games. This is what I will discuss first.

The moment that you start thinking about a game that you want to program, there is planning involved. I have never programmed a game that I never did any planning for. Yes, sometimes games get started and planning happens simultaneously, but in any case planning does happen. This is actually fairly critical for a cool game. Here is what has to be accounted for. First the setting and objective, both of which could determine the other. Secondly are more details such as what must happen for the game be won or lost. After that, anything can happen. This just a long way of saying that if you are going to program, there has to be an objective to your efforts.

A very important requirement is the concept of controlling the game character or object. This is a definite must. Otherwise the game obviously would have no purpose other than something like a screen saver. The main ways for the user to get input to the computer is through the keyboard, joystick/pad, and/or the mouse. This can all be accomplished in QBasic. Some of the main keys to use are the up, down, left, and right arrow keys, the space bar, tab and others. It is possible to access almost every key on the keyboard, including differentiating between F1 and Alt+F1 or Ctrl+F1. For a single-player game this can be perfect. With 2 player games, there are some possible minor conflicts that could arise; I'll explain later. The joystick can sometimes be a problem because it has to be calibrated every time the program is run. This creates a need for a lot of bothersome code that half the time won't do much good. On the odd chance that the joystick does work on the programmer's computer, the program is less than likely to work on another computer unless the joystick being used is the same kind as the programmer's. The mouse, on the other hand, is a very flexible piece of equipment and the code used to access the mouse is almost always compatible with any other computer with a mouse.

These days, the most focused part of a game is the way that the player(s) can view what they are doing, usually through the graphics capabilities. The method for doing this is through a point of view. There are only so many ways of displaying what's going on but the most common points-of-view for games written in QBasic are top view, side view, or a combination of both. Even then there is the options of the orientation of the screen relative to the viewer. There are different referred ways of doing this when comparing one and two player games, so that will be discussed in the separate sections.

Sometimes the best part of a game is its title page. This does help create some interest for the player(s) the first time they see the game. A cool logo might make the difference between someone trying your game or trying someone else's. They say not to judge a book by its cover, but it still happens all the time.

ACTION GAMES: SINGLE-PLAYER

Planning

You, bad guys and a fight to the death; this is the main reasoning behind almost any single-player computer game. Fortunately, people have made millions by making this a little more interesting than that. There are thousands of possibilities for what the player can be, what the enemies are, and how the fight is carried out. Balls, blobs, people, tanks, boats, spaceships, futuristic psycho-mobiles… anything can be programmed into what could become a really cool computer game. This just goes to show that it really isn't hard to figure out what the setting for a single-player game is going to be.

Point of View

For each game that is written, one of the biggest decisions that is made is that of the point of view of the game. Is the player going to have a bird's-eye view, a side view, or a first person point of view. Regarding graphics or the equivalent, this determines the entire processing of the game. The first two points of view make for much easier programming than the third. There is not much involved (none for many games) in the way of programming 3d graphics calculations, which can cause a great disadvantage in the availability of processing speed, not to mention the difficulty in figuring out what equations and formulas to use. Of course, this is all assuming that you want to have the view of your character/object "look around" the playing area. If this is not the case, a first point of view is almost as simple as the others.

Screen Behavior

One thing that might greatly influence the decision on what point of view to use is the behavior of the screen. Is the screen going to scroll or remain static? An example of a scrolling screen would be a game with a space ship, viewed from above, that stays in the center of the screen with stars going by in the direction that the ship is pointing. An example of a static screen would be a car or something that moves around on the screen, trying not to run into things while shooting aliens. A combination of the point of view and the behavior of the screen is a part of game creation that should take considerable thought. It is very unwise to start programming a game that is too difficult to complete with much more experience than one has. It causes discouragement and it can become a bad habit of storing up lots of uncompleted games on your hard drive.

Once the point of view and screen behavior is chosen, the bulk of the game processes can be worked out. This includes starting to program in a few graphics such as borders and other things that you might want on the screen. Some things that are great for something like space ship games are gauges and indicator lights of sorts. However, a programmer can't get too far into this process before having to start actual programming of the game.

The Main Character/Object

If your program is going to be a side or top view, you'll need a character or object with which to "interact" with everything else in the game. This is your good-guy. This is where you hopefully have decided what to use for this role. If it is a box, ball, or car, it is easy enough to draw these objects and save them into arrays for easy access with the PUT command, even if it means saving four drawings of it, one in each direction. However, if a person is to be the character, things get anywhere from a little to very

complicated. (see STRATEGY GAMES) Whatever you decide for your character to be, it will still need to be drawn on the screen. Depending on the simplicity of your graphics this can be done in different ways. This I will discuss later on.

Enemies

In any game, there is always an opponent of some sort. Aliens, space ships, rocks, bullets, sometimes even the clock is working against the player. Hopefully, for every game the enemies and other objects will correspond to the setting and the main character/object of the game. For example, a spaceship and planets along with enemy space pods and weapons will work well together. Of course there is the bizarre game with a miniaturized car racing around a kitchen counter, proving that anything can be put together to a somewhat reasonable game. In any case, the character or objects used in a game should make a bit of sense together and should add to the excitement of the game.

There are a few things to consider in creating the graphics for the enemies. The way that the graphics for these characters or objects are drawn depends greatly on the point of view. If it is a side or top view, you need only to draw them in much the same manner that the player's character is. However, with a first point of view, the enemy might need to be drawn several different sizes to represent 3-dimensionality. In the section on programming structure I'll have more about drawing characters and objects for a game.

ACTION GAMES: MULTI-PLAYER

Planning

The main difference between a single-player game and a multi-player game is that instead of being against the computer, the players are against each other, unless of course the game is the sort where both players are on the same team against the computer. Both of these kinds of games sometimes require more planning than a single player game. The object is more easily defined: "get the other guy". The setting and playing characters may even be just as easy (or difficult) as with a single-player game. This section will explain what there is involved in creating a multi-player game.

Points of View and Screen Behavior

One of the main differences in options between a one-player game and a two-player game is how much weight is put on the decision of whether to use a certain screen behavior/structure and the point of view. The easiest is overhead-fixed screen view. The screen does not require scrolling (static) and both players can view everything at once. An example of a static screen is a game like Battlefield Annihilator (found at http://www.bitsmart.com/qbstation). For other screen behavior, adding the attributes of scrolling and other points of view make things rapidly increasingly difficult. A 3D map game, where each player has a 3D view from a first point of view, is not going to work very well in QBasic if you are expecting to achieve good quality graphics. Another difficult point of view, for any game really, is a side or top view that has a scrolling background which contains many colors/ pixels compared to the background color. For example, a side-scrolling game that has a man running along a downtown street with buildings and cars rushing by might work in QBasic but it wouldn't do it very fast. Anyway, enough about what is bad, on with the good...

Main Characters/Objects

Usually in games, each player begins with the same advantages. There are some games that have different advantages for each player that make the game more interesting, and that is a detail that can be worked out very easily, assuming that you have good programming habits. Whether there is a difference in advantages or not, the main point here is that there still should be a differentiation between the two players game pieces. A space fighting game should at least have the opposing players' ships be of different designs, even if the weapons and all are identical. In helping you decide what you want your characters to be, consider the following.

Since the game is entirely dependent on conflict, there should be a good reason for it. Aliens and men, germs and antibodies, missiles and flares, any pairs of things that tend to conflict with one another would be acceptable, at least as long as the two "teams", so to speak, are approximately equal. A distinct inequality will weaken the game's ability to interest people and will be discarded.

Enemies

Yes, even in a multi-player game there can be enemies, in addition to the other player. There can be a common danger such as an earthquake or volcano, falling off a cliff, getting zapped by lightning, being hit by a comet or asteroid, or being grabbed and eaten by a sea monster. Common enemies and/or dangers can be an excellent addition to multi-player games. However, though it might enhance the excitement of the game, if the enemy is too perilous, the game will be too hard trying to fend off the secondary target while ignoring the primary target-the other player.

Key Usage

Many two-player games have both players using the keyboard at the same time. This has its advantages and disadvantages. (As I promised previously, here is more detailed discussion on this matter.) There is an advantage of somewhat interchangeability between players if programming has been done correctly. Two similar "maps" of keys on the keyboard can be made, so that anyone can use either side of the keyboard with little difficulty. Using a mouse or joystick for one player and the keyboard for another, there would be much more conflict in changing places, and this is the kind of conflict to avoid since it can distract from the game and become frustrating. On the keyboard, though, if one player holds a key down, expecting it to have a continuous effect just before the other player presses a key even once, there can be confusion and the controls seem unresponsive. The solution for this is to press the key over and over. This is less that ideal for several kinds of games, but it is not that big of a problem for most. Players who can't control themselves not to cheat may cause another problem. If one player is using the arrow keys and the other is using other parts of the keyboard for similar use, the way to disable the arrow keys for the first player is simply by holding down the Ctrl key. This disables all the keys, of course, but this toggle can easily be switched on and off. This can actually be done from either player. The Shift keys can change any number keys that are being used to irrelevant inputs. I haven't found all the problems, but even still, this is something that can easily be remedied by fair play.

It might have seemed a little confusing as to why it is important for someone to be able to play both sides of the game. This is mainly because if there are two different people who play one the same side all the time play together, there is conflict because neither one knows how to control the other player. It's just one of these few things that can gain brownie points towards the popularity of a game.

Common Items

There are some things programmed into games all over the place that help each player fight against the other. Here is a short list of some of those things:

Radar

For multi-player games there will probably be some element of having an advantaged viewpoint of the other player. This is a fancy way of saying radar. A radar, probably as the most common piece of "equipment" on games, I would say is a standard issue in games. With radar, each player knows where the other is, at least in a general area. If you have a game that has split-scrolling screens and the characters, objects move around a large playing area, a radar will increase the amount of contact and conflict between the two players, also eliminating time consuming searching which would put a negative effect on the game overall.

Gauges

If there is going to be a competition of power, which there usually is, there has to be a way of displaying this for all to see. A player is not usually going to attack another without knowing that the differences in power availability is going to be his/her advantage, right? The remedy for this is a simple gauge in the form of a line or circle or even a digital display, however you want, just as long as it gets the information across. Gauges do determine a significant part of a game's results, so be sure to include them where appropriate.

Indicators

If there is a selection of weapons, how is the player supposed to know which on is selected, it wouldn't be good to waste firing a precious, heavy-duty missile at something that would take only a simple bullet to destroy. If there were "buttons" of sorts that could be highlighted, it would be easy to tell what is selected. This goes for anything that might be created for the sake of saving key usage as referred to previously. Simple lights might be good for some purposes too. An indication that an enemy is in range for firing upon would be very useful. Or on the other side of things, a flashing warning light could tell the player that he/she is in danger of being killed or destroyed. One indicator that is sometimes overlooked is the score that the players have achieved, if applicable. This may be in points or in the form of a clock. Games that have the attributes of racing should always have a clock to show how well (or how badly) someone is doing.

Targeting Devices

A targeting device can be designed in many ways. There is the simple top view method that draws a circle around the opponent when in range. Another way is making the aiming "X" controlled by the player to jump to the center of the target as long as the aim of the gun or whatever weapon being used is aimed approximately in the right direction. A good idea would be to have the aiming indicator be drawn around the selected target and be able to rotate between different targets. There are many ways to program targeting systems that fit properly in a game. If this is to be a feature in a your game, consider the advantages and disadvantages of each method and choose the kind that will enhance your game the most.

A FEW SPECIFIC SUGGESTIONS FOR ACTION GAMES

There are mounds of ideas that could improve the quality of games programmed in QBasic, and I've found quite a few of them. Some are simple and easy to see, others are so obvious they might have been missed. Some just came from years of experience.

Two Player Keyboard Layouts

When designing a two-player game it is imperative that there is a way to control the game, otherwise there is no purpose, as I stated before. The trick in using the keyboard for two players is, of course, making sure that there are no keys that are common between the two players. The Space Bar is definitely one to leave out, for example. It is universally known as the shoot key, but not so for single-computer multi-player games.

This is something that I have used a few times for multi-player games. If a game does not require than 10 or 12 keys per player each player can have a kind of copy of the controls, one on each side of the keyboard. The arrow keys correspond with S, X, Z and C, and the keys above the arrow keys, Ins, Del, Home, End, Page Up, and Page Down correspond with 1, Q, 2, W, 3 and E. That makes 10 keys for each player. Need a few more? The Backspace and Enter keys are good doubles for Tilde and Tab keys. If there is still a shortage the F-keys can be accessed. F1 to F4 go along with F9 to F12, although the placement relative to the rest of the players' keys isn't quite as interchangeable and some adjustment must be made in switching sides. In the case of these extra keys, this is not too critical of a problem.

Graphics Related Ideas

There is an endless possibility of things to do with graphics. Unfortunately, only a small fraction can be done in QBasic, especially with a slower computer. A general rule to follow is that of keeping graphics simple enough so that the frame rate of the game cannot be noticed.

Line Drawings

Many graphics games that are considered cool are based on line drawings. This is not the best quality of graphics, but it does work. Line drawings focus more on keeping the program running as fast as possible, but also delivers a decent quality appearance. If we are talking about top-view game spaceships, the DRAW command using TA (turn angle) can be used to draw a good looking ship which can then be erased then drawn again at a new angle. This does let the background show though, but PAINTing in the center of the ship will give it some body and look not too bad, especially if you have the fill in color differ from the outline color.

GET and PUT

These are the shortcuts for animated graphics. A pre-drawn picture can be stored in RAM as an array using GET and then can instantly be accessed through PUT. If you have a rather complex graphic that must move around on the screen but would take much too long in the execution cycle to redraw each time it moved, PUT is the solution. Drawing the graphic(s) beforehand and staring them in an array can speed up the process incredibly. “PUT (x, y), arrayname, OR” will instantly draw the graphic which is in the specified array to the screen at the coordinates indicated. All that has to be

done then is to replace the background, if any, and PUT the graphics in it's new location. The OR attribute indicates to QBasic that the graphic is to be superimposed on the background.

Just for reference, replacing a complicated background can be done in much the same way that the graphic is put on the screen. Simply save into a temporary array the area that the graphic is to be drawn over. When the graphic should be erased, PUT the saved background over top of the graphic, being sure to use the OR attribute as indicated above. This then gets rid of the graphic from its old place on the screen and replaces it with the original background. This will run into problems, though, if any part of your background is COLOR 0.

Times of No Action

When there is no movement on the screen, it is not necessary for the unchanging graphics to be continually be drawn on the screen as long as nothing is affecting them. For example, if an object is no longer moving on the screen, it might be reasonable to skip the erasing and redrawing of its graphic as long as there is no movement or other effect such as another graphic moving over it. This can be done by a simple comparison that makes sure the x and y velocities of the object are both 0, then, if they are, skip the drawing and erasing subroutine. NOTE: Be sure that the input used to control the ship is NOT in the drawing routine. Otherwise, if the object that is being controlled stops, it's won't be able to start again.

CONCLUSION

Now that you have read this entire tutorial, you have barely scratched the surface of the gaming industry, congrats. The other 90% comes from experience. The number of failed and incomplete games that I have on my computer is embarrassing, and through them I have learned a few things about what to do and what not to do. I hope that this tutorial has given you a few good ideas and some worthy guidelines.

Written by Jack Thomson for
The QBasic Station
http://www.digital-forces.com

Please send questions/comments to: qbstation@hotmail.com