Daniel Davies sketch extension: Space Invaders

For this week's assignment, I took the opportunity of using SDL to create my own version of the classic 1970s arcade game "space invaders". The game really uses a fairly simple concept: use your "space ship" to shoot bullets at oncoming space aliens to stop them from reaching your zone.

My program works in the same way as traditional space invaders. If you do decide to have a go, there are 3 simple control buttons for the game:

-The space bar is used to shoot a bullet
-The left arrow key is used to move left
-The right arrow key is used to move right

The game ends either if the aliens reach your zone, or if you manage to destroy all the aliens. A score was also provided as a means of allowing the user to see how well they did (unless they win the game, at which point this isn't necessary, as it's the "max" score).

Compiling:

I decided to also create a Makefile along with this game, as I felt they would be useful in this circumstance. There are 3 separate cases in my version of space invaders: the user can either run autotesting, or they can choose to run space invaders in 2 different levels- easy or hard. These two levels simply differ by how fast the aliens move, and hence how much time the player has to shoot them down. To run these different cases, the user first must compile, and then to run autotesting, the user can type: "./invade", to run easy mode, the user must type "./invade e", to run hard mode, the user must type "./invade h". Or, in the make file structure:

Simply typing "make" will run autotesting on the game
Typing "make e" will run the game in easy mode
Typing "make h" will run the game in hard mode

The program

I followed your advice on updating the screen in games: so the entire screen is refreshed at once, rather than refreshing specific parts of the screen depending on different moves/ times etc.

After deciding how the screen was going to be refreshed, it was fairly simple to figure out that at any one point, there could be potentially be three objects moving on the screen at any one point:

-the bullet
-the aliens
-the player

Bullet

The bullet was programmed to be identical to that in the real version of space invaders. A bullet can be fired by the player at any point in the game, as long as there isn't already a bullet on the screen. As for the speed of the bullet, I first tried to observe how fast the bullet in real space invaders moves, and then imitate this- with the constraints that the bullet is fast enough to enable a user to hit a target in a short amount of time, but also not too fast as to make the game too easy. In the end,

moving the bullet at roughly 1/3 of the size of the ship (roughly 10 units) every 50 milliseconds gave an optimum speed.

Aliens

The aliens move at a rate of X units across every half a second- X depending on which difficulty the user chooses, either 40 or 20 for hard and easy respectfully. The aliens move just like in the original game- left or right on the screen, and if at the edge of the screen, also downwards by 50 units. The game ends as soon as the aliens move into the same row, or "zone", as where the player is located.

Player

The player was made roughly the same size as an alien. On each click of the arrow keys, the player moves 25 units across the screen (apart from when the player's ship is at the edge of the screen of course: at this point the ship doesn't move if the user tries to move it out of the screen). As in the original game, the player can move any time in the game.

Drawing the characters

I did originally plan, as you advised, to use images for the characters on the screen. I researched methods to do this, and tried to get it working (using methods such as SDL_LoadBMP and SDL_BlitSurface), but unfortunately couldn't get it to work. Because of time constraints on this project, I unfortunately had to move on to complete the mainframe of the project by using your line functions to draw out characters. In the end this actually turned out to be fairly advantageous, as the characters could be drawn with relatively good detail to resemble what they look like in the original game, and it was also very flexible to use when moving these characters around the screen.

Drawing the player with lines was relatively simple, whereas drawing the frame of the alien required quite a lot of lines: in the end, I decided to put all drawings of a line into one reasonably big function. Although this looks large, I recall you saying that it may sometimes not be worth splitting a large function (such as testing for example) if it is not logically complex/ follows the same pattern, which it exactly what this function is: simply numerous calls to the function that draws a line on the SDL window. I could have easily split this up into separate functions, but this felt like it would have been for the sake of splitting it up, and would have made it harder to follow. So I instead labelled what each group of calls to the line function does with parts of the alien, separated by spaces in the lines, just like testing. If this was the wrong decision, I would happily like to hear how I could improve it and what I should have done.

Main Game Frame

To use SDL in this project I used the functions you provided for sketch, but with some extra modifications for extra features. This includes certain SDL function I found on google (such as replacing the SDL_WaitEvent function with the SDL_PollEvent function to avoid halting the game while checking for a keypress event) and also other modifications to change colouring etc.

Before the game starts, the user is also greeted with a terminal message and a countdown. To make this possible, the newDisplay function was split up into 2 sections.

What counts as a hit

Generally speaking, the alien is considered shot if the bullet strikes it at any point. This includes the bottom of the alien (most cases), but the program also accounts for the fact that aliens may “run into” the bullet on a sidewards motion, as in the original game.

Autotesting

Autotesting was, as always, provided to test all key functions in the program. Autotesting includes 50 tests.

Conclusion

I actually really enjoyed making my first real, fully functional game in SDL and like the challenges it brought. It’s a shame at how time constrained the projects are, as there are many extra features that could be added to the game to make it better. This is something I will definitely pursue in my own time. If possible, a topic I would find interesting for a tutorial session at some point could be focused on how these original games functioned on limited computing power and graphics.