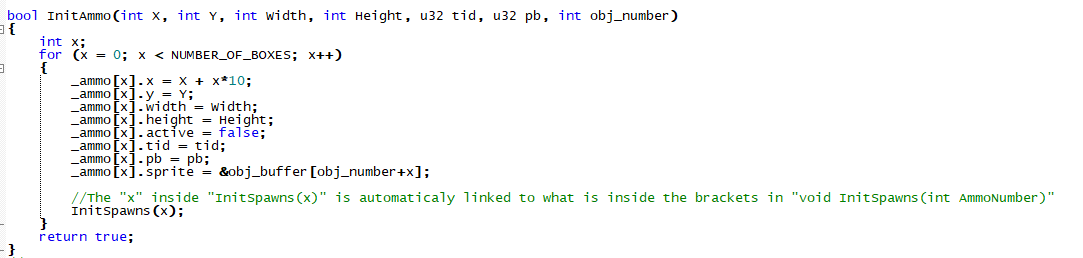


**Remarks**

In order to create a function that sort our scores in a descending order, we created a third Integer “temp”.Then we made a for loop that check our score (the increasing of the score in the same function for collision between ammo box and the player) against every high scores. Finally here we get to the core of the function, the if statement, if the actual score of the player is bigger than the high score,move the high score’s value in the int temp (for don’t get it overwrite by the score) them move the score in the top scores,and at the end we move the int temp in score,so if is it bigger than one of the high scores it’ll gonna re-trigger the if statement till the score is not less than the top scores.

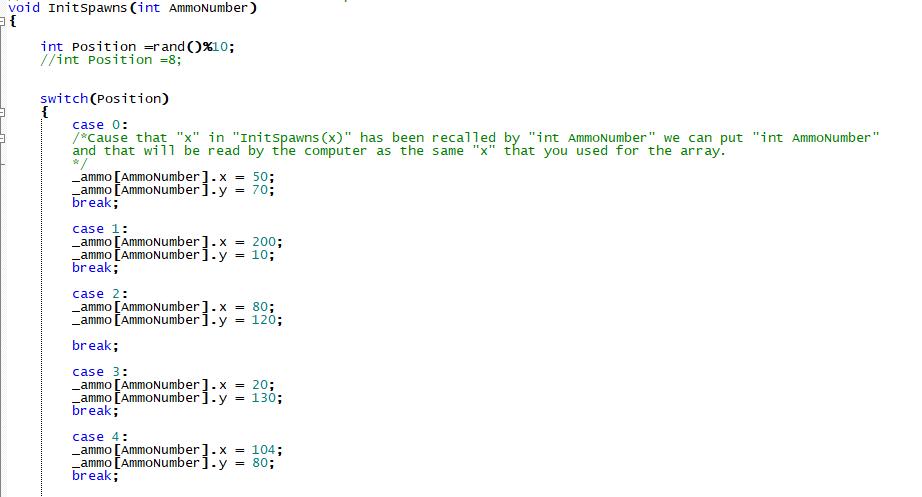
|  |  |
| --- | --- |
| Routine | Required header |
| CalculateScores | LDD.h |



**Remarks**

Just initialising some data

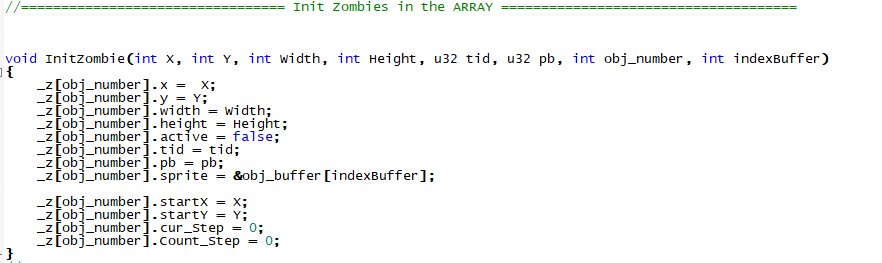
|  |  |
| --- | --- |
| Routine | Required header |
| CalculateScores | LDD.h |



**Remarks**

Here we initialise the various spawn locations for our ammo boxes (that now look like cakes),here we use as well “and()%10” that is related to the int FrmCount (for count the frames as substitute of time). As we can read from the comments what is inside the brackets of the function , “int AmmoNumber” is just a kind of pseudonym for the “x” in our previous function, this is very useful cause by changing the” #define AMMO\_OF\_BOXES” we automatically change the “x” and “AmmoNumber”.

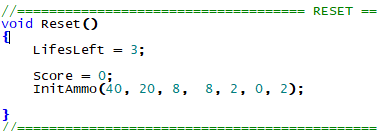
|  |  |
| --- | --- |
| Routine | Required header |
| InitSpawns | LDD.h |



**Remarks**

Here we initialise the Zombies.In this case we got 3 interesting elements: “…].cur\_Step”,”…].Count\_Step” and “indexBuffer”.The first one is for check the direction,the second one is the number of steps before we change the direction.The “indexBuffer” is for allocate every zombies in its space in the memory,we use this value for the zombies and not for the ammo cause the ammo has been initialise all together but each zombie got a different path to follow,so we need initialise them one by one,as is shown in the file “function.c”.

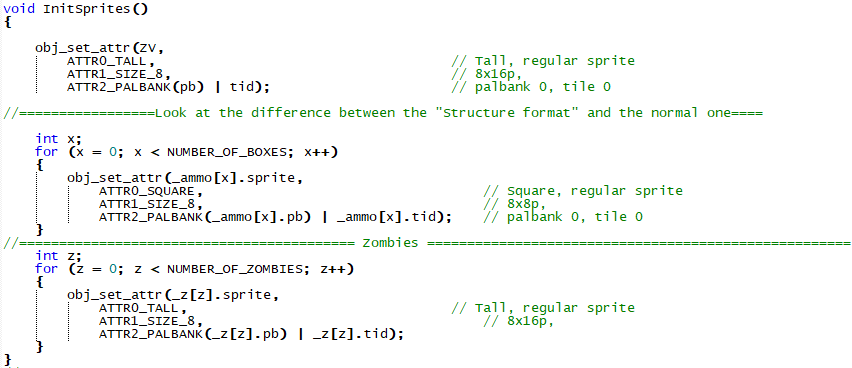
|  |  |
| --- | --- |
| Routine | Required header |
| InitZombie | LDD.h |



**Remarks**

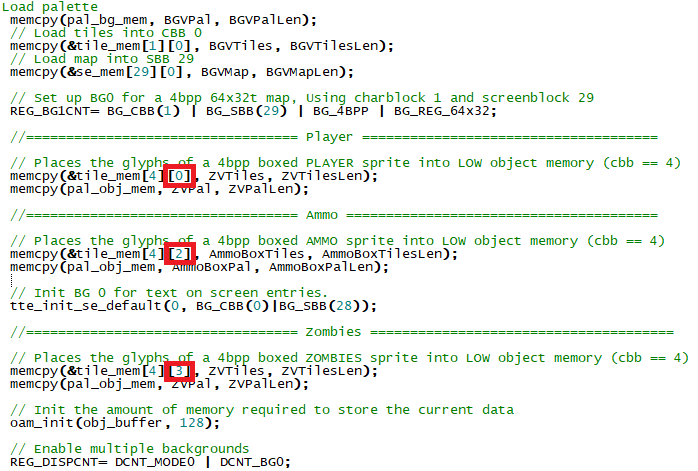
Here we are resetting some values and objects for allow the users to replay the game.

|  |  |
| --- | --- |
| Routine | Required header |
| InitZombie | LDD.h |



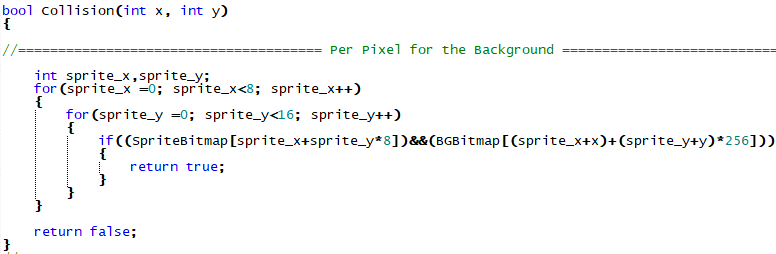
**Remarks**

Here we initialise the Sprites,that mean that we are bounding the graphics to these objects.



And here (we are now inside the main) we complete the bonding process,the red squares show the tile ID,that is what really bound the objects to their graphics.On top we can see also the setting of the Back Ground

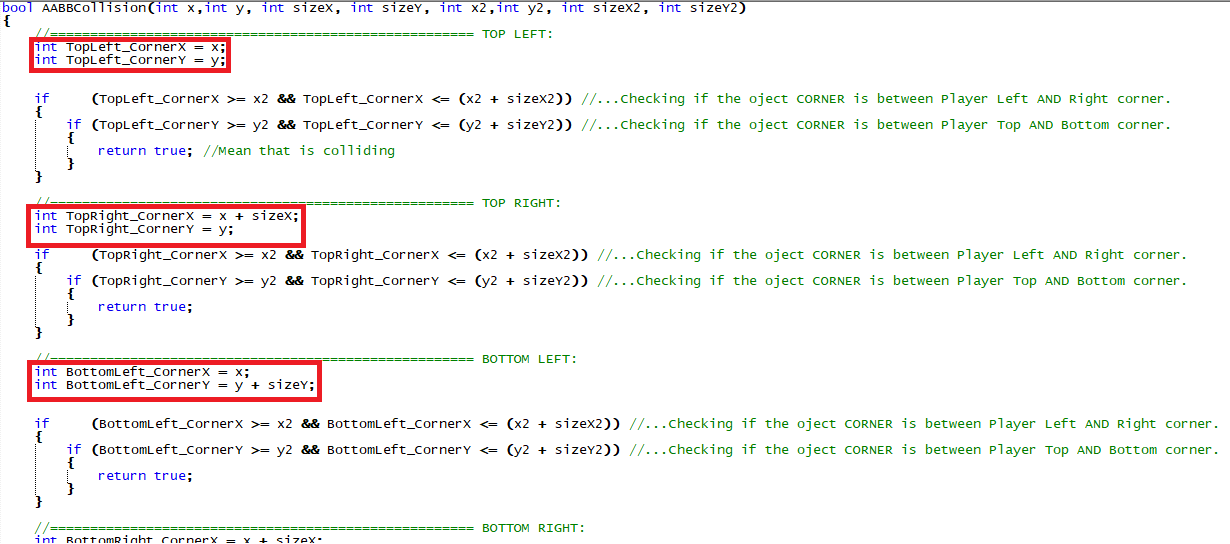
|  |  |
| --- | --- |
| Routine | Required header |
| InitSprites,main | LDD.h |



**Remarks**

But before we enter in main there are a two bools more to analyse, the first is this,the bool for the Pixel per Pixel collision. This a pixel collision that require a BITMAP,indeed it is checking between the player’s bitmap and the BackGround’s bitmap.By a double for loop (one that check the “x” and one the “y”) against the bitmaps.

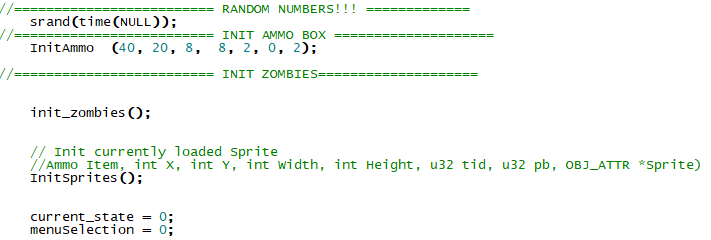
|  |  |
| --- | --- |
| Routine | Required header |
| Collision,Z,BG. | LDD.h,Z.h,BG.h |



**Remarks**

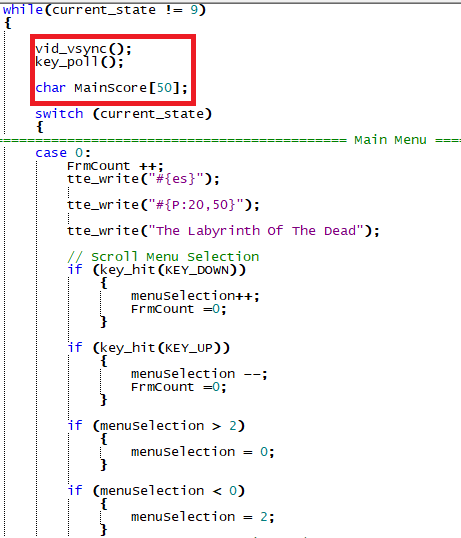
Here the last bool before we get to the “main”.Here the redefinitions in the red squares play a very important role in the code-understanding. All the if statement are checking if one corner is over the left top corner or between that cited corner and his width, the opposite and the same for the length.

|  |  |
| --- | --- |
| Routine | Required header |
| AABBCollision | LDD.h |



**Remarks**

Before we enter in the state machine there are some recalls and the setting of some data. On top we can see the last bit that allow to the random numbers to work. At the bottom we got the “current\_state” that is for check the actual game state, and the “menu selection” that is for switch the various selections in the menus (basically allow to a little arrow to appear in front of our menu selection).



**Remarks**

Here we gonna analyse just some salient features of the state machine. On top we can see in the red square in descending order: the refreshing code for the screen, the code for read the inputs, and the char that is for show int etc.

Underneath of “case 0:” we can see “FrmCount ++” that is our system for count the seconds.