# Boost Tasks

Bonus lives – Getting 1,000 points will net you an extra life, up to a maximum of five. Clearing all the main enemies in a level will get you 1010 points, and hitting the UFO will grant you a large point bonus, so you are guaranteed at least one extra life per completed level.

Audio – All the original game’s sound effects are present (enemy movement, player shots, impact explosions, and UFO), though I did make the UFO sound a little less grating on the ears! I have used the FMOD API to implement this, as it has multi-channel sound playback, but there is also the option to use Windows’ built-in sound handler by changing one boolean in GameData.

Advanced AI – Only the bottom-most enemy of each column will shoot at the player – can’t have the enemies shooting through each other, can we? Also, each enemy can only have one laser on-screen at a time, so as to avoid the player getting caught in an inescapable onslaught on enemy fire. In addition, the enemies will always move all the way to the edge of the screen – if a column on the end is taken out, they won’t count towards the group’s movement, since they are dead. The UFO will also grant the player either 100, 150 or 200 points, at random, upon being shot.

High Scores – Just like the arcade original, your high score will be saved and carried through game-overs and new levels, always displayed at the top of the screen. Also just like the original, the score will be lost if you shut down the game.

Animation – The enemies do the classic Space Invader Wiggle™as they move, and explosions each have two frames of animation. The enemies’ lasers also have their own Wiggle™ manoeuvre, provided they are the rarer zig-zag shaped ones.

# Post-mortem

## The Good

I have created, to the best of my ability, a faithful recreation of the arcade game. Every detail, from the AI patterns, to the number of points given an enemy is defeated, has been thoroughly researched and implemented. I have kept a well-maintained codebase with consistent naming schemes and a coherent class structure. I have also completed the boost tasks to flesh out the game a bit more.

## The Bad

The commenting in my code could be improved. As it stands, if I were to hand my code off to another programmer, they would have to spend a while figuring out the ins and outs of my code. There are also a small number of ‘hacky’ elements that go against OOP – most notably in the enemy manager’s collision checking functions. While these functions are not heavily coupled to the type of object being passed in (they do have some default behaviour if an unknown object type collides), the fact that the enemy manager handles almost all collisions isn’t very encapsulated. Ideally, the enemy manager would give Game a list of all the enemies and bullets that need collisions checked, and Game would run through them itself.

## Extra Details

I am proud of the way I worked around integer sprite coordinates – by using a float array for actual distance calculations, and then rounding it to an integer for placing the sprite in the world. This makes the game more compatible with processors of different speeds, since sprite positions can be calculated using delta time. Delta time normally doesn’t work so well with integer values – if it is less than 1, then it would get rounded to 0 and the sprite wouldn’t move at all.