Space Invaders

Peter Hamilton and Jared Pilcher

Fall 2011

Contents

1 Space Invaders Overview **3**
- 1.1 History of Space Invaders . . . 3
- 1.2 Game Play . . . 4
  - 1.2.1 Objective . . . 4
  - 1.2.2 The Tank . . . 4
  - 1.2.3 Enemies . . . 4
  - 1.2.4 Points . . . 5
- 1.3 Game Details and Specifications . . . 5
  - 1.3.1 Aliens . . . 5
  - 1.3.2 Alien Mothership . . . 6
  - 1.3.3 The Tank . . . 6
  - 1.3.4 Bullets . . . 7
  - 1.3.5 The Bunkers . . . 8
  - 1.3.6 Scoring . . . 8
  - 1.3.7 End of Game . . . 8
  - 1.3.8 Appearance . . . 9

2 Game Console and Engine **10**
- 2.1 Game Console . . . 10
  - 2.1.1 Xilinx University Program Board . . . 10
  - 2.1.2 Xilinx Virtex II. Pro EDK System . . . 10
- 2.2 Game Engine . . . 10
  - 2.2.1 Software Game Engine . . . 10
  - 2.2.2 Software APIs . . . 14
  - 2.2.3 Meeting the Game Specifications . . . 14
  - 2.2.4 The Bunkers . . . 19
  - 2.2.5 End of Game . . . 19
  - 2.2.6 Appearance . . . 20

3 Bug Reports and File Organization **24**
3.1 Bug Reports 24
3.1.1 Drawing Bullets 24
3.1.2 Destroying Aliens 24
3.1.3 Moving Aliens to Right of Screen 25
3.1.4 Overlay of Bullet Black Boxes 25
3.1.5 One Alien Firing Two Bullets Simultaneously 25
3.1.6 Overwriting Memory 25

Chapter 1

Space Invaders Overview

1.1 History of Space Invaders

Designed by Tomohiro Nishikado in 1978, Space Invaders is one of the greatest arcade games of all time. It took Nishikado over a year to develope the game and its hardware. It was manufactured in Japan by Taito. It was so influential that it pushed the video game industry into a worldwide industry. Shortly after its release, it caused a shortage of 100-yen in Japan. By 2007, Taito earned over $500 million dollars in profits.

Space invaders was a really popular game in 1978. It is one of the most influential and successful games of all time. It is a two dimensional shooting game involving a tank that shoots aliens marching across a screen. When all of the aliens were destroyed, or all of the lives of the tank were used, the game ended. It has had several sequels, and was released on several platforms. It was the inspiration for many other video games on the market today.

To this day, a bug still exists from the original game. In the original game, as the marching aliens were killed, the game refreshed faster since there was less to load an the screen. This added an extra challenge to the game. As the game became popular, consumers became accustomed to the bug. When the developer fixed the bug, the market responded poorly. It was placed back into the game, and has been there since.

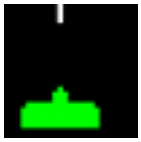
1.2 Game Play

1.2.1 Objective

The world is being invaded by marching aliens. All of humanity has gathered behind 4 bunkers for protection. You control the only weapons available, a supply of 3 tanks and a handful of missiles. Kill all the aliens! Watch out! The alien mothership flies over head! Destroy it at all costs! The fate of the world rests in your hands...

Win the game by killing all of the marching aliens. Destroying the mothership gives you extra points.

1.2.2 The Tank



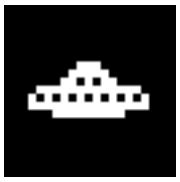
You are the tank. Push the left button to move left, and the right button to move right. You can fire and move simultaneously.

Watch out! Enemy fire rains from above! Dodge the alien fire by moving the tank left or right. Each time your tank is hit by an alien bullet, it explodes. You have three lives. You must survive...

Fire your missile to destroy the alien invaders! Push the middle button to fire your missile, aiming for the aliens. When the missile hits an alien, it dies. Be careful! They move left and right, and get closer as time goes on.

1.2.3 Enemies

Spaceship



The alien mothership circles the Earth, looking for its prey. Destroy it at all costs! It flies occasionally from left to right. As it flies overhead, launch your tank missile by pushing the middle button. Be careful, it tries to evade you missile, so shoot accurately and quick!

Aliens

55 aliens are marching toward your position! You are charged with destroying all of them before they reach you! Fire your missiles to kill them! These aliens can shoot up to four bullets at a time.

Aliens are capable of destroying walls, so your bunkers will provide no protection when they reach them. Their bullets slowly degrade your bunkers when they're hit. Be quick! You will soon have no protection!

1.2.4 Points

Earn points by destroying the aliens with your tank missiles. The lower two rows of aliens, middle two rows of aliens, and the top row of aliens are worth 10, 20 points and 40 points per alien, respectively.

Earn extra points for destroying the mothership! Earn between 50 to 350 points for each mothership destroyed.

1.3 Game Details and Specifications

1.3.1 Aliens

Alien Placement and Appearance

There are 55 aliens that start on the screen. They are on a 11 by 5 grid. Each grid is 15 pixels by 15 pixels. There are three types of aliens. The first

two rows are the widest aliens, the next two rows are a little narrower and the top row has the narrowest alien.

Alien Movement

The aliens move laterally in increments of 2 pixels. When the rightmost aliens reach the right edge of the screen, the grid of aliens will move down 7 pixels and start moving left. When the leftmost aliens reach the left edge of the screen, they will move down 7 pixels and start moving right. The rightmost alien and leftmost alien is the closest alien to the respective sides that has not been killed. This means that when the rightmost column of aliens are destroyed, the aliens march toward the side of the screen until the next column hits that side. The aliens continue this pattern of movement until the bottommost row of alive aliens reaches the bottom of the bunkers. The aliens will speed up in their movement as aliens are killed.

Alien Explosions

When an alien is hit by a bullet from the tank, it blows up. The other aliens continue onward while the explosion sequence is happening.

1.3.2 Alien Mothership

The alien mothership flies across the screen from left to right. It's time of appearance is random. When a bullet strikes the mothership, it flashes, revealing the score received.

1.3.3 The Tank

Movement

The tank moves laterally across the bottom of the screen. The user is able to push the right button to move right, or push the left button to move left at a rate of two pixels. The user is able to hold down the left or right buttons to keep the tank moving across the screen.

Firing Bullets

You may also fire a missile by pushing the middle button. The tank is capable of moving and firing simultaneously. It is also capable of changing direction while shooting. This means that you are able to hold down the fire button and change directions.

Death

When the tank is struck by an alien bullet, it explodes. The player will lose a life and continue the current level.

1.3.4 Bullets

Tank Bullets

Tanks are capable of firing bullets by pushing the middle button. Tank bullets, unlike Alien Bullets, only have one appearance. They are white rectangles that are ejected just above the center of the tank. No black boxes surround the bullet. In other words, when it leaves the tank, or strikes an object, an overlay of a black box cannot be seen.

When a bullet is on the screen, the tank cannot fire again until the bullet has left the screen. Each bullet travels at a rate of 2 pixels. When it strikes an alien or mothership, the object is destroyed.

Alien Bullets

There are two types of alien bullets:

1. Zig zag bullets rotate cyclically through the following appearances:

   (a) 

   (b) 

   (c) 

   (d) 

2. Cross bullets oscillate through the following appearance:

   (a) 

   (b) 

   (c) 

Each bullet is ejected from the center of an alien, randomly chosen to be an alien on the bottommost row. There can only be 4 bullets on the screen at any given point in time. Once a bullet leaves the screen, a new bullet can be fired. Each bullet travels at a rate of 2 pixels. When the bullet strikes a tank, it is destroyed. When the bullet strikes a bunker, the bunker is eroded. When a piece of the bunker is completely eroded, the bullet can pass through it. No black boxes surround the bullet. In other words, when it leaves an alien, or strikes an object, an overlay of a black box cannot be seen.

1.3.5 The Bunkers

There are four bunkers available for the tank to use as cover. Each bunker is divided into 10 sections. Each section partially erodes with each collision of an alien bullet. After 4 collisions, the section is fully eroded and will allow bullets to pass through.

When the aliens reach the bunkers, they will walk above the bunkers as they move through them. In other words, they will be drawn over the bunkers and those portions of the bunkers will be destroyed.

1.3.6 Scoring

Earn points by destroying the aliens with your tank missiles. The lower two rows of aliens, middle two rows of aliens, and the top row of aliens are worth 10, 20 points and 40 points per alien, respectively.

1.3.7 End of Game

The game ends when one of the following three events occur:

1. The bottommost row of aliens that are alive reach the bottom of the bunkers.

2. The tank has been destroyed three times.

3. All aliens are destroyed.

When the game ends, a game over screen is displayed.

1.3.8 Appearance

The game appearance is identical to the flash version of Space Invasion. The game runs smoothly and at a decent speed. The screen is also updated close to the same rate as the flash-based game. No artifacts appear on the screen. No flickering occurs on the screen.

Chapter 2

Game Console and Engine

2.1 Game Console

2.1.1 Xilinx University Program Board

2.1.2 Xilinx Virtex II. Pro EDK System

2.2 Game Engine

2.2.1 Software Game Engine

The structure of the Space Invaders Game is complex and intricate. The main() function initializes the state of all objects of the game. When complete, it enters an infinite loop, calling functions whose timers have expired. Timing is kept using the PIT and its handler. Using these timers, the tank, mothership, aliens, bullets, and explosions move at a configurable rate. As a bullet moves toward an object, its screen coordinates are consistently checked for a collision with a bunker, mothership, tank, or alien. When the object is struck, the states of the game are updated, and a collision is drawn to the screen.

The Space Invaders game uses double buffering. It draws only to the next frame, switches to that screen, and erases only the pixels of the objects that have changed in the previous frame. For example, when the tank moves, it's drawn to the next frame, the viewer sees the new tank, and the previous frame tank is erased in the background. This process is handled in our render() function, and occurs on regular intervals using the designed timers.

Figure 2.1 contains an overview of the structure of the Space Invaders Game. The red blocks represent the functions that form the skeleton of

the program. The green blocks represent the actions are are performed in those function. The yellow block represents the global array of implemented timers. The blue blocks represent the individual timers used to update the state of the game, or render the screen.

Timers

The sequence of events in the Space Invaders game is controlled using timers. After initializing the state of the game and creating timers, main() enters an infinite loop, determining when timer functions should be called. A timer is a structure that contains the interval of ticks between calling its registered callback function. Each time the PIT timer interrupt invokes the PIT handler, the global variable pit_counter is incremented. Because of the complexity of the game, many timers currently exist. These include:

- **New Space Ship:** If the mothership is not currently on the screen, this timed event has a random probability of initializing the mothership on the screen.
- **Move Aliens:** Moves all the aliens by a configurable amount of pixels. If the aliens reach the bottom of the screen, the Game Over state is initialized.
- **Move Space Ship:** If the mothership is currently on the screen, it moves by a configurable amounts of pixels across the screen.
- **Fire Alien Bullet:** If less than four alien bullets are currently on the screen, this timed event has a random probability of initializing a new alien bullet on the screen. A random, non-empty column among the aliens is chosen, and the bullet is fired from the lowest alien in that column.
- **Poll Buttons:** Checks the value of the push buttons. If the left button is pushed, the tank move left by a configurable amount of pixels. If the tank is already on the left edge of the screen, nothing is performed. If the right button is pushed, the tank moves right by the same amount. If the tank is already on the right edge of the screen, nothing is performed. If the middle button is pushed, a new tank bullet is fired from the tank. If there is already a bullet on the screen, nothing is performed.

- **Move Bullets:** All bullets move across the screen by their configured amount of pixels. Alien bullets move down the screen and tank bullets move up the screen. The location of each bullet is checked to detect possible collisions. See the section *Detecting Collisions* for more details.
- **Update Bullets:** Updates the stage of animation for each active alien bullet as they move across the screen.
- **Process Explosions:** Three types of explosions exist that exhibit different behavior. This timer function updates the animation of the explosion. When the animation sequence is fully performed, it is no longer drawn to the screen.
- **Render Screen:** Renders the screen. See the section *Rendering* for further details.

A PIT handler is used to increment the global variable pit_counter. After initializing the state of the game and creating timers, main() enters an infinite loop that performs the following:

1. A snapshot of pit_counter is taken.
2. This value is compared with the value of the previous snapshot.
3. It iterates through each timer and increments the timer's counter.
4. If the timer's counter exceeds the maximum value, it resets the counter and executes the timer's registered callback function.

The timer loop will continue to run until the game over state is reached.

Detecting Collisions

Figure 2.2.1 demonstrates the general logic used to determine a collision. Each time a bullet moves, its coordinates are compared against the coordinates of the bunkers, tank, aliens, and mothership. The collision detection algorithms are implemented in the detectCollision() function.

If the bullet is a tank bullet, its coordinates are compared against the coordinates of the bunkers, aliens, or mothership. The different collision scenarios are as follows:

- If it is within the coordinates of the bunkers, the collision detection function determines which bunker, and bunker block it hit. If the block is still active, the bunker is eroded and the bullet is deactivated.
- If it is within the coordinates of the aliens, the alien row and column are determined to specify the hit alien. If the alien is still active, the type of alien is determined. If the bullet is within the the bounds of that alien type, they alien explodes and the bullet is deactivated.
- If it is within the coordinates of the mothership, the mothership explodes and the bullet is deactivated.

If the bullet is an alien bullet, its coordinates are compared against the coordinates of the bunker, and tank.

- If it is within the coordinates of the bunkers, the collision detection function determines which bunker, and bunker block it hit. If the block is still active, the bunker is eroded and the bullet is deactivated.
- If it is within the coordinates of the tank, the tank explodes, and the bullet is deactivated.

Rendering

The screen is rendered on regular intervals by using a timer. The rendering process occurs whether or not the the state of the game is changed. This mirrors the rendering process of the flash game. This rendering process creates a very smooth animation process.

Double Buffering

Flickering on the screen is caused by drawing and erasing objects on the same screen. In order to avoid flickering, double buffering is used. Double buffering involves writing to another part of memory, or frame than the currently displayed screen. The currently displayed frame is never modified. After drawing is finished on this separate frame, the program changes the screen to display this new frame. Thus, all modifications to the screen occur on an inactive frame. Figure 2.2.1 shows the steps in rendering process.

Flying over the bunkers

Aliens must march over bunkers when they reach them. However, erasing and redrawing the aliens can erase blocks of bunkers, even though they aren't inactive. In order to redraw the bunkers properly, the state of the bunkers are always drawn in a dedicated frame, FRAME3. As an alien is erased, the program accesses the bunker pixel values that are stored in this frame to restore these pixels. When an alien is drawn in the bunker region, the background values of BLACK are replaced by the values stored in FRAME3. Thus the alien appears to march over the bunkers.

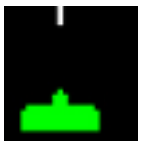
2.2.2 Software APIs

Rendering of images

Images are stored in memory as an encoded bitstring. A Xuint16 can represent a single line of an image up to 16 pixels wide. Each image is an array of these encoded bitstrings. To draw to the screen, the images are scaled up to double their size. A single pixel in the stored image corresponds to a 2 by 2 block of pixels on the screen. Each image type has it's own rendering function. The functions typically take an x and y coordinate, a frame to draw to, and a color to draw.

2.2.3 Meeting the Game Specifications

The Tank



The buttons are polled in regular intervals using the the Poll Buttons timer. As the tank moves laterally, it is drawn to a new frame, in a new position. The screen switches to that frame, and the tank in the old frame is erased, creating a smooth animation. Furthermore, the tank can only move to the edge of the screen. When it reaches the edge of the screen, it stops, unable to move left. Each time the tank moves, its coordinates are compared against the edges of the screen. When it reaches the edge of the screen, the moveTank() function doesn't change the coordinate of the tank.

The user is able to hold down the left or right buttons to keep the tank moving across the screen. This is done by using a timer to control the speed of the tank movement. The Poll Buttons timer is checked in regular

intervals. If a left or right button is asserted, it is update the location of the tank, moving left or right.

When an alien bullet hits the tank, the tank explodes. This is implemented using three tank animations that are stored in an array. Each time the explosion update timer is called, the animation changes to a new position in the array. This happens on regular intervals when an explosion occurs. The tank appears to explode because of the implemented double buffering and switching between these items in the array. A new animation is drawn to the new frame, while the old animation in the old frame is erased.

Each time a tank explodes, the lives global variable is decremented and one tank is erased from the screen on the top right corner.

Mothership

The mothership is randomly activated on regular intervals, flying from left to right. This means that after a certain period of time, the program determines whether or not the mothership should appear. The probability of its appearance is configurable.

When the mothership explodes, a random integer between 50 and 350 flashes for one second in its place. This score is added to the total score and displayed in the top left corner of the screen. The flashing is done by alternating between two animations in regular intervals. One animation shows the score of the hit, and the other animation displays black. Thus the score appears to flash.

Aliens

There are 55 aliens total that are formed in an 11 by 5 grid. Each alien is drawn in a 30 by 30 pixel block. Only the coordinates of the top left corner

of the top left alien is stored. All other aliens reference this value. An array 55 integers is kept globally to store whether or not the alien is active. Only aliens that are active are redrawn to the screen. The first two rows of are the widest aliens, the next two rows are a little narrower and the top row has the narrowest alien.

An alien becomes inactive when a bullet hits it. A hit occurs when at least one of the pixels of the bullet overlap the pixels of an alien. Each row of aliens is different. Although the aliens are drawn in blocks, unless the bullet overlaps a white pixel of an alien, it will not explode. This is necessary for a bullet to fly between aliens. For example, a bullet can miss a bottom row alien, and hit a second row alien. When an alien is killed, an explosion replaces it for a short time before dissipating. The other aliens continue onward while the explosion occurs.

Aliens march over bunkers when they reach them. However, erasing and redrawing the aliens can erase blocks of bunkers, even though they aren't inactive. In order to redraw the bunkers properly, the state of the bunkers are always drawn in a dedicated frame, FRAME3. As an alien is erased, the program accesses the bunker pixel values that are stored in this frame to restore these pixels. When an alien is drawn in the bunker region, the background values of BLACK are replaced by the values stored in FRAME3. Thus the alien appears to march over the bunkers.

The aliens move laterally in increments of 4 pixels. When the rightmost aliens reach the right edge of the screen, the grid of aliens move down 7 pixels and start moving left. When the leftmost aliens reach the left edge of the screen, they move down 7 pixels and start moving right. The rightmost alien and leftmost alien is the closest alien to the respective sides that has not been killed. This means that when the rightmost column of aliens are destroyed, the aliens march toward the side of the screen until the next column hits that side. The aliens continue this pattern of movement until the bottommost row of alive aliens reaches the bottom of the screen.

We check for the rightmost or leftmost active alien by iterating each alien in each column. If there isn't an active alien in a column, the next inner column is checked until an active alien is discovered. When we move the aliens, we do not change the direction of the grid of aliens until the rightmost or leftmost alien reaches the edge of the screen.

Each time an alien is killed, the animation speed of the aliens increases logarithmically. This is done using the timers. Each time an alien is destroyed, the time between alien movements decreases by 10

Scoring

Points are earned when the mothership is destroyed, or an alien is killed. This happens when at least one pixel of bullet overlaps a drawn pixel of an alien or mothership. The lower two rows of aliens, middle two rows of aliens, and the top row of aliens are worth 10, 20 points and 40 points per alien, respectively. When the mothership is destroyed, a value between 50 to 350 points is added to the total score.

The score is displayed in the top left corner of the screen. All numbers are left justified, imitating the flash game. Commas are drawn slightly lower than numbers. Thus, they appear to be in the correct position.

Tank Bullets

Tank bullets are fired when the middle button is pushed. The Poll Buttons timer checks the middle button on regular intervals for changes. When it detects that the middle button has been pushed, a bullet is drawn to the screen just above the center of the tank, and the global tank bullet variable is set to active. This is done by checking both the middle, left and right buttons every time the buttons are polled. Each button handler changes the state of the game before the next render. Thus the tank can move, change directions and fire between two calls to render(). Then, on regular intervals, the bullet moves upward until it leaves the screen or hits the mothership or an alien. When it strikes any of these two objects, it is deactivated, and it is neither drawn to the screen or move. Explosion sequences are activated when the tank bullet hits them.

When a bullet is on the screen, more tank bullets are not created until the middle button is pushed again, and the tank bullet is off the screen. Each bullet travels at a configurable rate.

Alien Bullets

There are two types of alien bullets:

1. Zig zag bullets rotate cyclically through the following appearances:

   (a) 

   (b) 

   (c) 

   (d) 

2. Cross bullets oscillate through the following appearance:

   (a) 

   (b) 

   (c) 

Each of these animation sequences are stored in an array of encoded bits. The location of the bullet determines which animation is drawn to the screen using modulus. Thus, as the alien bullets move downward, they cycle through the animations, making them appear to zig zag or oscillate.

The bullet regularly moves using a timer by a configurable amount of pixels. Each time the timer calls the moveAliens() callback function, the bullets position is updated, and compared with the position of the bunkers and tanks. If any of the bullet's pixels overlap any of the pixels in these objects, an explosion occurs, and the bullet is deactivated.

Each bullet is ejected from the center of an alien, randomly chosen to be the bottommost alien of an active column. This means that when a bottom row alien is destroyed, the alien just above it is able to fire a bullet. We implemented this by randomly choosing a column of aliens. Iterating upward, we searched for the first active alien in that column. When it was found, the alien bullet used the coordinates determine where to be drawn..

Only 4 alien bullets are active at a time. In our code, this means that there are only 4 alien bullet structures. Once a bullet is active, it cannot be reactivated until it strikes a bunker or tank, or leaves the screen. The movement of the alien bullet determines if it destroyed an object. If its new coordinates cause it to overlap an active bunker block or tank, it causes them to erode or explode.

No black boxes surround the alien bullets. In other words, when it leaves an alien, or strikes an object, an overlay of a black box cannot be seen. This is done by Only drawing only the foreground color of the bullets.

2.2.4 The Bunkers

When a tank or alien bullet strikes a bunker, the bunker erodes. This is done by dividing the bunker into 12 equal size blocks. An array of 48 integers controls the state of each block. When a block is hit by a bullet, it's associated integer is incremented until it reaches 4. When it reaches this state, bullets can pass through them, and are not destroyed when they hit them.

A function strategically places the bunker block in the correct positions of the screen. Since they do not move, they are always drawn in the same position.

2.2.5 End of Game

The game ends when one of the following three events occur:

1. The bottommost row of aliens that are alive reach the bottom of the bunkers.
2. The tank has been destroyed three times.
3. All aliens are destroyed.

When the game ends, a game over screen is displayed.

First, when the aliens move, if their position is is equal to the position of the bottom of the bunkers the game_over global variable is set to 1. Second, when the tank is destroyed, if no other lives exist, the game_over global variable is set to 1. Third, when an alien is destroyed, if not other aliens are active, the game_over global variable is set to 1.

In the main() function, when the game_over global variable is set to 1, it stops checking the timer values and jumps out of the while() loop. The aliens are erased, and the game freezes with a large GAME OVER text in the center of the screen.

2.2.6 Appearance

The game appearance is identical to the flash version of Space Invasion. The game runs smoothly and at a decent speed. The screen is also updated close to the same rate as the flash-based game. No artifacts appear on the screen. No flickering occurs on the screen.

The render() function makes this possible. Because the screen is updated on small intervals, the screen is consistently updated close to the rate of the flash-based game. This eliminates flickering, making the animation smooth.

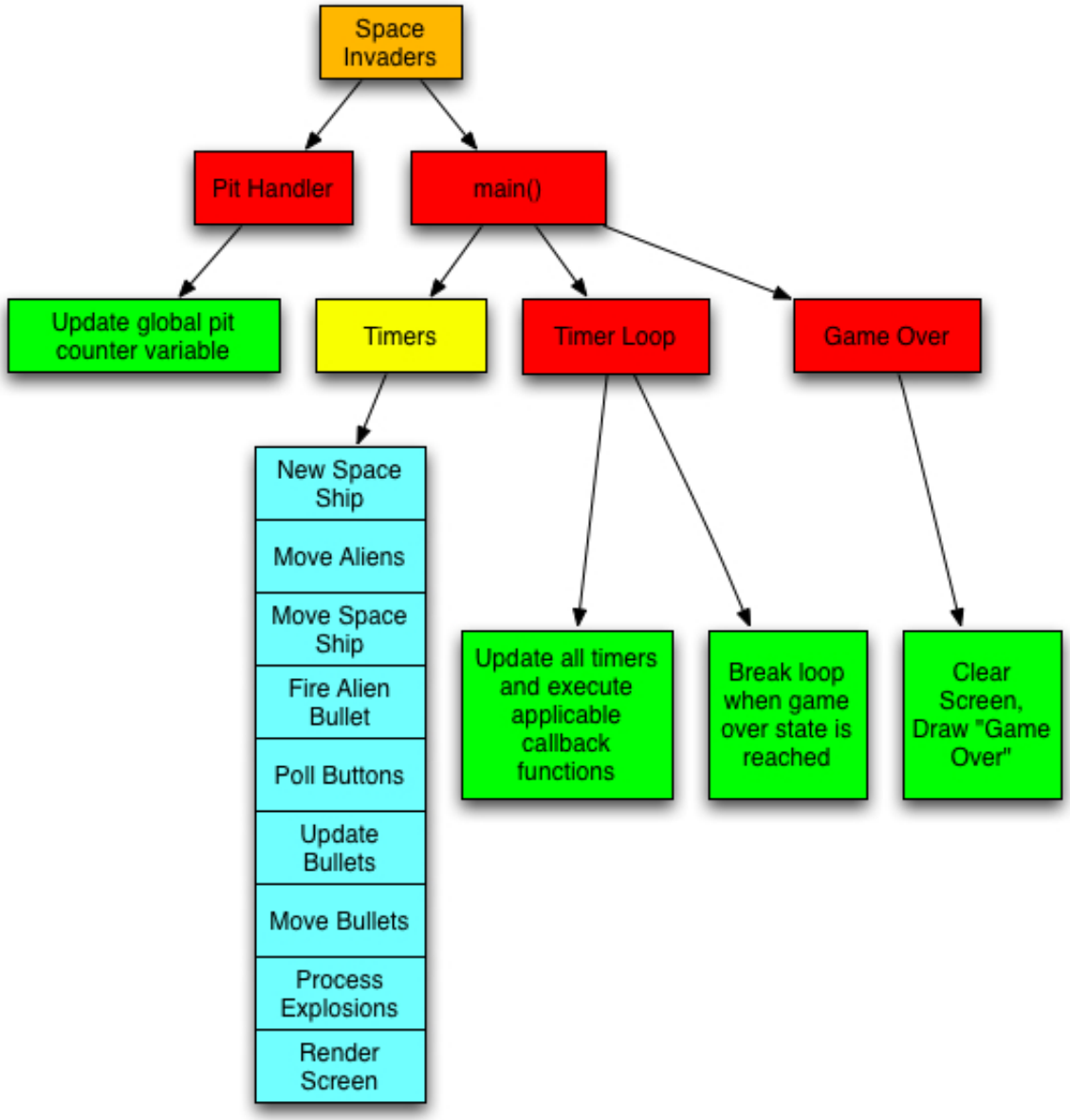


Figure 2.1: Space Invaders Overview. The red blocks represent the functions that form the skeleton of the program. The green blocks represent the actions are are performed in those function. The yellow block represents the global array of implemented timers. The blue blocks represent the individual timers used to update the state of the game, or render the screen.

Figure 2.2: Collision Detection. The green blocks represent the state that must be met to continue. The orange blocks represent the process of determining the specific object that was hit. The blue blocks represent the actions taken to change the game state during a collision.

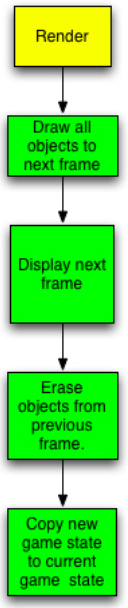


Figure 2.3: Double Buffer. The currently displayed frame is never modified. Only after writing correct values to a new frame, the program changes the screen to display it.

Chapter 3

Bug Reports and File Organization

3.1 Bug Reports

3.1.1 Drawing Bullets

During our design phase, we found a bug involving the drawing of alien bullets to the screen.

First, we had difficulty cycling through the different different stages of the bullet. We were able to cycle through four stages, however, the fourth state was a different bullet type entirely. We fixed the bug by placing a simple if statement in our draw bullet loop. When the bullet was in the fourth stage, it cycled back to the third state to and reset its state to the first state. In this way we were able to use the same functions for a four cycle bullet and a three cycle bullet.

Second, we found a bug when the first bullet was fired. Without permission, another bullet was drawn at pixel location (0,0). We found the problem by analyzing our bullet structs. We discovered that our bullet variables were not initialized to zero. We assumed that they would be initialized to zero, but a random value was placed there. When we initialized all variables, the bug was fixed.

3.1.2 Destroying Aliens

During our design phase, we found a bug involving the destroying of aliens. We thought that the lab instructions said to give find a random number between 0 and 54 and destroy that alien. When we tried to pass it off, the

TA told us that we needed to ask the user to input a number. To fix the bug, we used the read() function to acquire a number from the user, one integer at a time. We destroyed the alien that corresponded to that number.

3.1.3 Moving Aliens to Right of Screen

We initially had some naive logic concerning the aliens reaching the edge of the screen. We set a hard limit for the grid of aliens. Once the top left corner of the grid had reached the limit, the aliens moved down and changed directions. This did not account for all aliens on the edge of the grid being dead, which would require the alien grid to move one column closer to the edge than it did with all aliens alive. We fixed this by using a more dynamic approach.

3.1.4 Overlay of Bullet Black Boxes

During our design phase, we found a bug involving the overlay of the bullet black boxes. We discovered that as a bullet was launched, and flew over objects on the screen, it had black boxes surrounding it. We have not yet resolved this bug, but will do so in the near future.

3.1.5 One Alien Firing Two Bullets Simultaneously

While trying test our implementation of aliens firing bullets, we encountered a situation in which an alien fires two bullets on top of each other. They will appear as one bullet. We solved this by limiting the firing of a bullet until the first one has moved out of its original position.

3.1.6 Overwriting Memory

With good intent, we wrote a for loop to iterate through an array, and ended it using a call to sizeof() in our main() function. Our screen exhibited strange behavior, drawing objects in the top left corner of the screen. Ensuring that we initialized our objects to be inactive, they still appeared there. Using print statements throughout the program, we determined that the initial values of our variables were incorrect. They seemed to change value within only a small block of initialization code in main(). Determined to discover the issue, we closely scrutinized our code and discovered the issue. We learned that sizeof() returns the number of bytes of the entire array, instead of the number of objects in the array. After correcting this mistake, our issue was resolved.