Approach to Change Update

Changes to this ("Approach to Change") document are shown in blue text.

Again, we separated our new documentation and code from the team we inherited from; this technique worked well in our third assessment and so we did not see a reason to change our approach to this aspect. However, we somewhat changed the manner in which we did it; during assessment 3, we used Team Muscovy's website files for reference for most of the project and only uploaded them to our own website at the end of the assessment duration. However, this approach had risks; as we have no control over the other team, they may (inadvertently or purposely) change the files or URLs contained on their website, which could render some of our documentation incorrect or inconsistent. Therefore, we uploaded Team Smew's documentation to our own team website at the beginning of this assessment. We then knew that we could accurately refer to URLs and files throughout.

A reason that we kept our new documentation and software separate was to allow better tracing of changes; it was clear and easy to see where changes were made when it was possible to compare side-by-side with the original. Github was very useful in helping us to retain the original code, along with previous code before commits were made. This also allowed for a fall-back working implementation in case any changes introduced new bugs, or files became lost/corrupted, etc.

As for how this technique helped us assess our requirements fulfilment – the traceability was, as before, useful for this, but in a slightly different way. In our previous swap assessment, the game was considered partially complete; therefore, some requirements were completely fulfilled, others half-fulfilled, and others not fulfilled at all. This meant that traceability allowed us to keep track of how each different case was being treated and which requirements still needed work to fulfil. However, this time around, all the previous requirements were complete. Our changes were implemented with regards to our own added requirements which were completely new; this method, therefore, let us see with distinction the difference between our inherited, 'complete' game, and our altered game. This allowed us to clearly refer to how our changes fulfilled the Assessment 4 brief.

Our team's approach to actual implementation of changes used the 'change-request' format, with minor changes. By 'change-request' format, I am referring to the handling of change whereby personnel involved with the program may request (propose) changes [1]. In a larger-scale, typical industry project, there are more personnel which may propose changes - for example, users, designers, system analysts, and so on [1]. The requested change is then evaluated by a Change Control Board - in this case, our team as a whole - based on cost, and benefit to the project. Following this, it is either accepted or rejected, and given a priority [1]. The process is shown in diagram form in Fig 3 [1]. As a smaller-scale project, we handled change proposal itself slightly differently - we didn't feel the need for a change proposal form, or similar, due to the tight nature of our team. Instead, changes were proposed through the 'Issues' functionality in github.

Previously, our code changes (issues) were organised by category – requirement fulfilling, bug-fixing and enhancement, in order of urgency. However, for this assessment, virtually every change was requirement-fulfilling. As we had inherited a complete game, bug-fixing and enhancement was no longer necessary. While the proposal of changes helped us to clarify what needed doing, and to label completed and incomplete changes as such, each change was about as important as another. All were also mandatory. Later on in implementation, enhancements were suggested based on the team's response to *how* the added requirements had been fulfilled, but the initial focus was on completing the mandatory changes. We also began using the 'checklist' feature in our github issues, which allowed us to organise wide changes like 'Demented mode' into sub-changes in a checklist. We then ticked them off when there were done, making our progress extremely clear and easy to see.

This also meant that we would no longer have to implement deletive changes in the code, and perfective and corrective changes were highly reduced in scale. For very minor perfective and corrective changes, we used the same method as last time of allowing team members to carry them out on their own discretion. Additive changes were given the highest priority, as the requirements changes given in Assessment 4 were additive themselves. Therefore, these changes were the most important to adhere to this assessment's brief.

Documentation changes were handled with different tools, as the issues functionality refers to Github code. However, the process by which they were carried out was essentially the same. Changes were instead proposed through our communication platform, slack. The ease of communication and use of Google Drive to share files made flexible change management like this convenient. Google drive was also useful as it allowed other members of the team to proofread partially-completed documentation and suggest improvements.

Requirements and Architecture Changes

Requirements

Our requirements changes came directly from the Assessment 4 briefing information, in which the required software changes were detailed. These are all additive changes, as previously discussed; this meant that the originally inherited requirements, which had been fulfilled in full by the previous team, did not require altering. Instead, we added new requirements as needed; these requirements fleshed out and specified how our interpretation of the Assessment 4 brief was to be implemented. We maintained the requirements format and organisation we inherited from the previous team, as it was clearly set-out and thorough. The changes discussed in this report are organised with regards to this format.

Requirements Section: Gameplay Systems

Two new requirements (with sub-requirements) were added to this section:

G14: "Demented AI behaviour mode: AI-controlled enemies will have a chance to spawn or become demented every frame. While they are demented, they will randomly pick one of four demented behaviours".

G14 specified the requirement of a 'Demented Waterfowl' mode (Assessment 4 Requirements Changes, http://www-module.cs.york.ac.uk/sepr/RequirementsChanges-201516.html) for the **AI-controlled** enemies (requirement **G8**). Justification for this requirement was that adding a chance of becoming demented to computer-controlled enemies fulfils the requirement of demented computer controlled waterfowl. The fact that computer-controlled AI may spawn demented at a small probability at each frame fulfils the specification of adding unpredictability to AI-controlled waterfowl's behaviour. The fact that, when demented, the computer-controlled AI will exhibit one out of four illogical behaviour, picked at random, implements the demented AI's random behaviour. The change can be seen in our updated requirements document at http://teal-duck.github.io/teal-duck/Assessment%204/Req4.pdf.

G14.1: "Demented waterfowl behaviours are picked at random from:

- Stand still
- Walk north
- Attack nearest enemy
- Run away from player"

G14.1 is a follow-on requirement we added to G14; it details the choices of behaviours demented waterfowl pick at random when their demented mode is triggered. These behaviours were chosen to fulfil the 'demented' requirement, as they are all illogical behaviours for an enemy entity to exhibit. Again, the random nature in which the behaviour for each entity is determined contributes to the 'random and unpredictable' behaviour required in the Assessment 4 Changes (http://www-module.cs.york.ac.uk/sepr/RequirementsChanges-201516.html). This change can be seen in our updated requirements document at http://teal-duck.github.io/teal-duck/Assessment%204/Req4.pdf.

G14.2: "Demented waterfowl become cured after 20 seconds of being demented"

G14.2 is the second follow-on requirement to G14. It specifies that demented waterfowl become cured of demented mode after 20 seconds of being demented. This was added because of the requirement in the Assessment 4 brief that demented entities should be able to return to their normal behaviour somehow after becoming demented (http://www-module.cs.york.ac.uk/sepr/RequirementsChanges-201516.html). This requirement ensures that no AI-controlled waterfowl will stay demented forever. This requirement change can be seen at http://teal-duck.github.io/teal-duck/Assessment%204/Req4.pdf.

G15: "There should be two available cheats. These cheats should enhance the player-controlled character, but should not make gameplay too easy or unfair"

G15 specified the requirement for 2 distinct cheats to be available in the game, which enhance the player character's skills. These cheats should not make gameplay too easy or unfair which is also mentioned in the requirement as it was specified in the Assessment 4 brief - as a "game-breaking" cheat would lessen the immersion and engagement level of our game, we made sure our requirement adhered to this point specifically. This change can be seen at http://teal-duck.github.io/teal-duck/Assessment%204/Req4.pdf.

G15.1: "Rapider fire: This cheat will make the standard fire rate as fast as rapid fire, and rapid fire's fire rate faster"
Specification of our first cheat variation. We felt that this cheat would make the gameplay easier to an extent, but not so much so that the game would not be engaging. It would also make getting the rapid fire pickup even more rewarding as the player's fire rate would increase again. It also alters the normal game state in favour of the player, and therefore is adequate to be considered a cheat. Therefore we felt this requirement was appropriate as a response to the Assessment 4 brief. The change can be seen at http://teal-duck.github.io/teal-duck/Assessment%204/Req4.pdf.

G15.2: "Infinite flight: This cheat will make the player able to fly for an unlimited amount of time"
Specification of our second cheat variation. We felt that this cheat, which allows the player to have an unlimited movement speed boost, would not make the game too easy or less fun. It also alters the normal game mode in favour of the player, and therefore is adequate to be considered a cheat. So we felt this requirement was also appropriate as our other implemented cheat. The change can be seen at http://teal-duck.github.io/teal-duck/Assessment%204/Req4.pdf.

G15.3: "Cheats should be toggled on and off through a cheats screen"
G15.3 is a follow-on requirement from G15 and specifies how the player will activate the two available cheats. We decided to make the cheats toggle on and off through a cheat screen. This allows the player to turn on/off the cheats whenever they like, and also makes it easy to test that the cheats are functioning correctly. This change to the requirements can be seen at http://teal-duck.github.io/teal-duck/Assessment%204/Req4.pdf.

Requirements Section: Control/Movement
A requirement plus its sub-requirement were added to this section:
C6: "Demented Player mode: continually and after a period of time the player character will contradict a user input when this mode is activated"
C6 specifies the requirement of a 'Demented Waterfowl' mode (Assessment 4 Requirements Changes, http://www-module.cs.york.ac.uk/sepr/RequirementsChanges-201516.html) for the **player-controlled character**. Justification for this requirement is that it came directly from the scenario: "For a demented player duck, they must randomly contradict an instruction over a period of time". While this gave us less freedom, it produced a highly specific requirement borne directly from the client brief. Therefore C6 is a simple, clear requirement, and can be seen at http://teal-duck.github.io/teal-duck/Assessment%204/Req4.pdf.

C6.1: "The player is cured of demented mode after 20 seconds of being demented"
C6.1 is a follow-on requirement to C6. It specifies that the player-controlled character becomes cured of demented mode after 20 seconds of being demented. Assessment 4's Requirements Changes (http://www-module.cs.york.ac.uk/sepr/RequirementsChanges-201516.html) said that there must be a method by which demented waterfowl become cured. This requirement fulfils that specification for the player-controlled character. This requirement change can be seen at http://teal-duck.github.io/teal-duck/Assessment%204/Req4.pdf.

Architecture
There were not many significant architectural changes implemented; this is because we built our changes upon the functionality and architecture we were provided with by the previous team. This meant that in almost all cases our changes were implemented by simply adding methods and attributes to already-defined class and inheritance structures. However, the cheat screen was implemented by adding a new class; the 'CheatScreen' class, inheriting from BaseScreen.

Software Changes

Code

Implementing G14-G14.2

The code for spawning demented mobs (**G14)** was implemented in the 'Round' class. This did not require any big architectural change, but required altering of the 'spawnRandomMobs' method. A chance of 10% for a demented melee mob spawning was added, and a chance of 2% for a demented ranged mob spawning was added to the method. This logic can be seen at https://github.com/teal-duck/mallard/blob/master/core/src/com/superduckinvaders/game/Round.java.

Mob's have a new boolean **demented** (in the 'Mob' class) which is true if they are demented, and false if they are not. Logic was also added into the Mob class to allow AI enemies to have a small chance of becoming demented every frame (**G14**). This fulfilled the requirement of implementing a demented mode wherein the AI-controlled characters display demented, random and unpredictable behaviour (see Assessment 4 brief).

The logic for the enemies' behaviour selection was also implemented in this class (**G14, G14.1**). This was through the addition of an enum, **DementedMobBehaviour**, where the options are ATTACK_CLOSEST, WALK_NORTH, RUN_AWAY and STAND_STILL. A further function, randomBehaviour, was implemented - which returned an instance of this enum, chosen at random. This function provided the logic for a demented enemy to randomly choose which behaviour to exhibit.

Logic for the actual functionality of these different behaviours was also added to this class. The method newDementedEffect checks which enum the enemy's behaviour is associated with and controls the enemy's behaviour accordingly (**G14.1**). This ensured the implementation simulated random, unpredictable behaviour in the enemies.

Curing the AI-controlled waterfowl's demented mode was also implemented in this 'Mob' class (**G14.2**). The update method checks, at each delta, whether the computer-controlled enemy has reached its maximum demented time of 20 seconds, implemented through adding a MAX_DEMENTED_TIME variable to the class. If it has reached the maximum, the method clearDementedEffect() is called, which returns the enemy's behaviour to normal. This ensured that the requirement of allowing demented waterfowl to be cured of their demented mode was fulfilled.

We chose to implement the logic this way, using the Mob superclass, because the previous team(s) had already provided a solid inheritance model and opportunity for us to create the methods and attributes needed for the functionality with their implemented architecture. We had inherited a fairly well-made game and did not find it necessary to re-organise the class system, or anything of the like. For that reason, all these changes were made additively. All of the above logic and code discussed can be seen at https://github.com/teal-duck/mallard/blob/master/core/src/com/superduckinvaders/game/entity/mob/Mob.java.

Implementing G15-G15.3

The cheats (**G15, G15.1, G15.2**) were implemented with changes to the 'DuckGame' (https://github.com/teal-duck/mallard/blob/master/core/src/com/superduckinvaders/game/DuckGame.java) and 'Player' (https://github.com/teal-duck/mallard/blob/master/core/src/com/superduckinvaders/game/entity/Player.java) class. There were no major architectural changes involved with implementing the cheats functionality, but the 'update' method in the Player class was altered to provide it, along with boolean values added to the 'DuckGame' 'Session' method to signal whether each cheat was activated or not.

G15.1: Rapider Fire:

While rapider fire is toggled 'on' in the cheat screen, the player has standard rapid fire before acquiring the rapid fire pickup, and then doubly-quick rapid fire after acquiring the rapid fire pickup. This was implemented by:

- Adding a boolean variable, named **rapidCheat**, to the Session method of the DuckGame class.
- This boolean is set to false by default, but is toggled to true if the player enables the Rapider Fire cheat in the cheat screen.

- When true, the update method in the Player class adds an additional Rapid Fire multiplier to the player character's fire rate.
- Therefore, the player's standard and rapid fire rate are faster.

G15.2: Infinite Flight:
While infinite flight is toggled 'on' in the cheat screen, the player's flight timer never runs out. This was implemented by:

- Adding a boolean variable, named **infiniteFlight**, to the Session method of the DuckGame class.
- This boolean is set to false by default, but is toggled to true if the player enables the infinite flight cheat in the cheat screen.
- When true, the update method in the Player class does not decrease the player's remaining flight time after each change in time while the flight key is pressed.
- Therefore, the player can fly (a speed increase) for as long as they like when the cheat is active.

G15.3: Cheat Screen
The new cheat screen was implemented by creating the new class 'CheatScreen' that extends 'BaseScreen'. This involved creation of a new class with an inheritance relationship, so modified the architecture. As discussed previously, this class also required changes to the GUI. The purpose of this screen was to provide the player a way to toggle the two available cheats on and off.

Implementing C6-C6.1
These requirements, relating to the demented mode of the player controlled character, were fulfilled through changes to the 'Character' class and the 'Player' class, which inherits from Character.

The player becomes demented by being hit by a 'Projectile' object from a demented enemy mob. If this happens, the becomeDemented() method, which was defined in character (and called to override in Player), is called, which sets the character's new **isDemented** boolean variable to true. This fulfils the specification in the brief of adding a demented mode to the player character (see Assessment 4 Requirements Changes).

Then, after each small time period defined as the MAX_DEMENTED_BETWEEN_EFFECT_TIME variable in the Player class, the Player method applyDementedEffect is called. This inverts the player direction controls, therefore fulfilling the requirement to have the player ignore and contradict the user's inputs while demented (**C6**).

The player is cured of demented mode after 20 seconds of it being active. The maximum amount of time the player can be demented for is decided by the variable MAX_DEMENTED_EFFECT_TIME held in the Player class. Again, these changes were made using the class system already implemented by the previous team, which we felt was solid enough to be sufficient for our needs and we therefore left intact.

The method updateDementedTimers was added to the Player class; this method updates all timers at each frame. If the amount of time the player has been demented is equal or greater to the maximum time, the method stopDemented() is called an the player is cured of demented mode. This implements the requirements for waterfowl (including the AI and player-controlled) to become cured of demented mode (**C6.1**, Assessment 4 Requirements Changes). All of these changes as discussed can be seen at https://github.com/teal-duck/mallard/blob/master/core/src/com/superduckinvaders/game/entity/Player.java and https://github.com/teal-duck/mallard/blob/master/core/src/com/superduckinvaders/game/entity/Character.java.

GUI
Throughout the previous assessments, we have ensured that the GUIs we produce enhance the user experience of the game. As well as being aesthetically pleasing, it is important that the GUI reduces the cognitive load on the player as they play [2]; this makes their experience more enjoyable and engaging [2]. This can be done by simplifying the design, ensuring that **all** information required by the player is clearly displayed, and ensuring the GUI remains consistent and not confusing.

To do this, we made sure to modify the GUI so that the it is clear when the demented mode is active, and when it is not. As the demented mode has a significant impact on the behaviour of both the player-controlled character and the AI, it is vital that the player is notified of when it is active on each entity. It is

also important that this notification is obvious, easy-to-see and understandable. Therefore, we added a demented indicator (purple swirl) which displays above the head of any entity with the condition - this can be seen in Figure 1, 2 and 3 in the appendix. This indicator disappears when the character is no longer demented, therefore displaying an accurate picture of the game state to the player at all times. This allowed us to sufficiently implement the demented-mode requirements (**G14-G14.3, C6-C6.1**) while maintaining a suitable GUI.

The next change to the GUI is our inclusion of a Cheats screen. We kept the Cheats screen very minimal and simplistic to ensure it would not look cluttered, but would still be sufficient for its purpose. We chose to implement cheat toggling by tickboxes; this method is easy to understand and requires only a click to change. Methods of navigating away from the cheat screen and back to the menu is also provided. This GUI change can be seen in Figure 4 and 5 of the appendix. These screenshots show that the selection boxes display an 'X' symbol when selected, which clearly conveys to the user that the cheat is currently active. Similarly, the boxes are empty when the cheats have not been selected as active by the user.

Bibliography

[1] Enzhao Hu; Yang Liu, "IT Project Change Management," in *Computer Science and Computational Technology, 2008. ISCSCT '08. International Symposium on* , vol.1, no., pp.417-420, 20-22 Dec. 2008
[2] Fei Hu; Lixia Ji, "On the Peak-Experience in the Game GUI Design," in Management of e-Commerce and e-Government, 2008. ICMECG '08. International Conference on , vol., no., pp.204-207, 17-19 Oct. 2008

Appendix

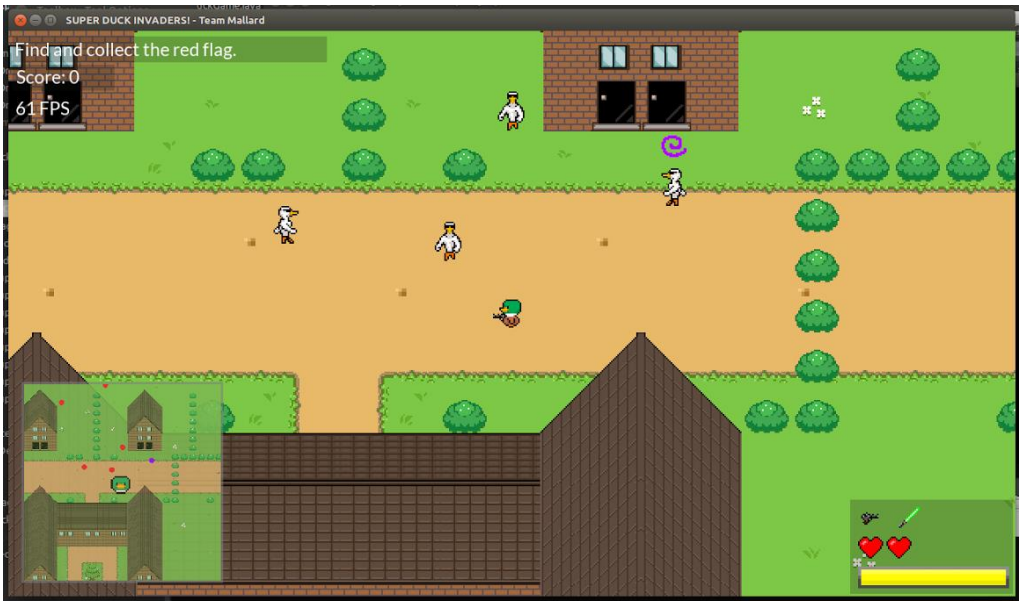

Figure 1; the game screen with a demented enemy shown, three non-demented enemy and the non-demented player

Figure 2; the game screen with a demented enemy and demented player on screen

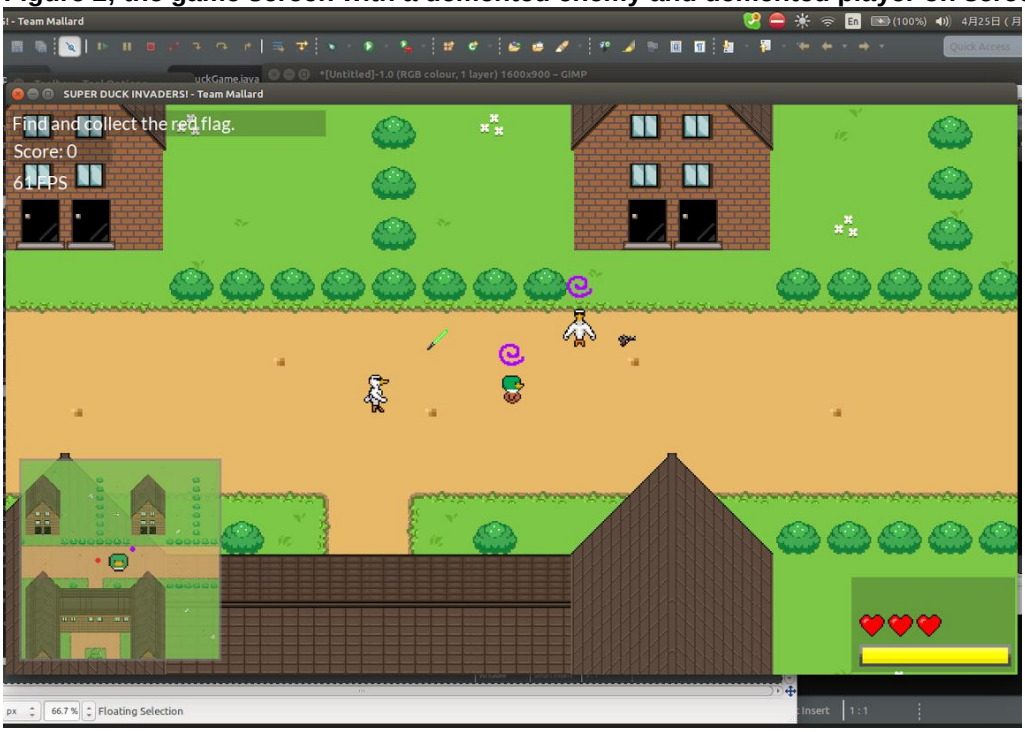

Figure 3; the game screen with demented player, demented ranged enemy and non-demented melee enemy

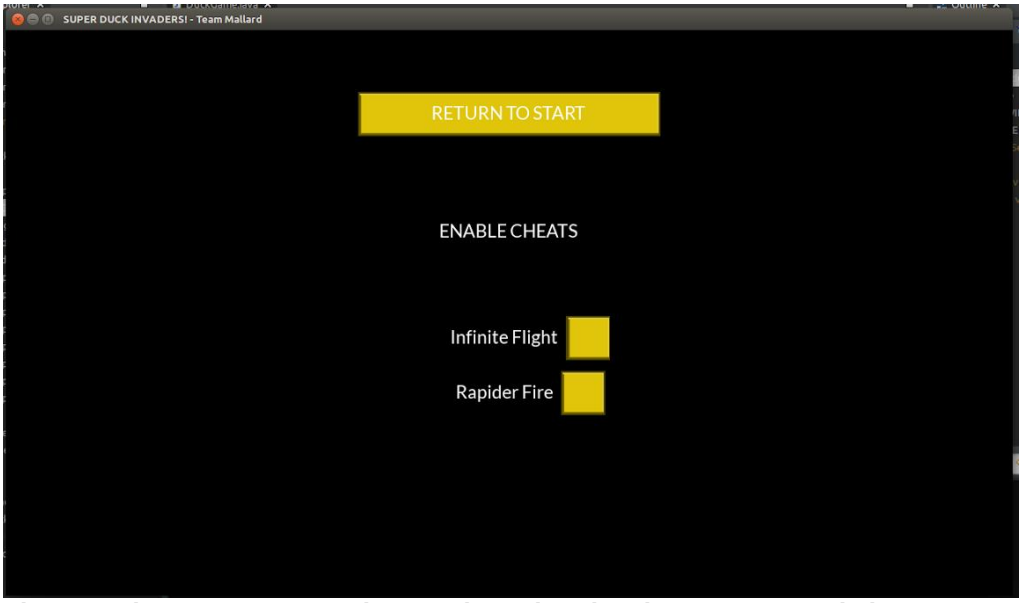

Figure 4; the game screen when navigated to the cheat screen and cheats are not selected

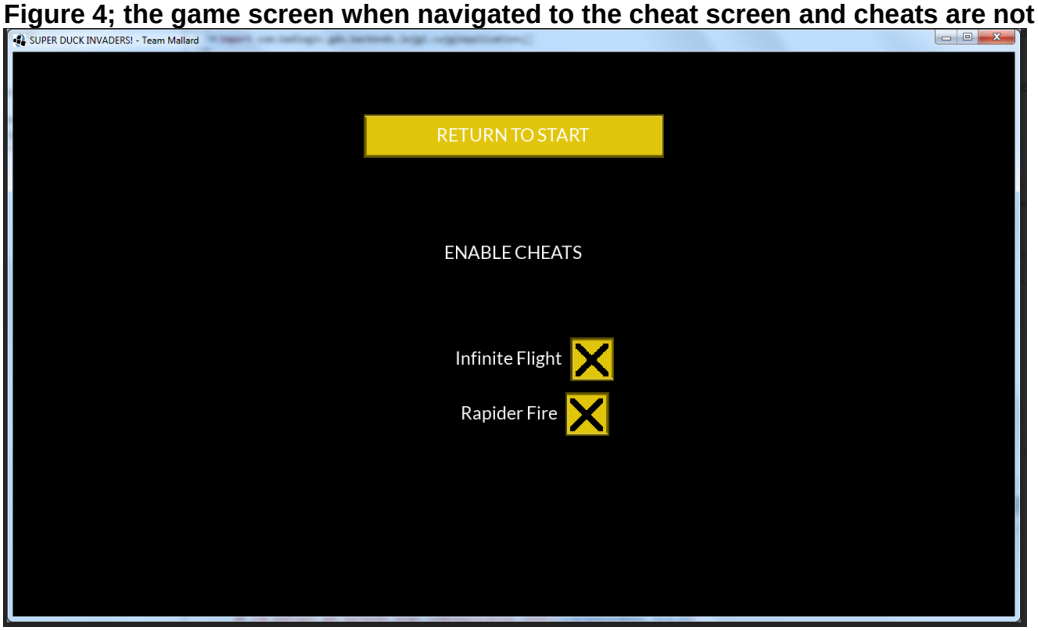

Figure 5; the game screen when cheats are both selected as active