Report
Duck Hunt Deluxe

Bachelor Computer Science

2023-2024

Table of Contents

Contents

1 Introduction

This document is an overview about phase 1 & 2 of the implementation of the NES game Duck Hunt in the programming language scheme (R5RS). This is the final product.

2 ADTs

2.0.1 Overview

- Game base structure.
  - Game ADT (Not really an ADT)
  - Position ADT
  - Draw ADT
- In-game objects
  - Duck ADT
  - Bomb ADT
  - Droplet ADT
  - Powerup ADT
- User-controlled
  - Gun ADT
- Stats
  - Score ADT
  - Lives ADT
  - Round ADT
- Helping ADTs
  - Hitbox ADT

The above text provides an overview of all the ADT's that have been implemented in the game. This version of the game contains all needed functionalities for phase 2. The final game includes an unfinished Game ADT, due to the complexity of the game it was impossible to organize.

2.1 Game Base Structure

2.1.1 Game ADT

As mentioned above, the Game ADT is not an actual ADT, it contains the bulk of the code of the game and allows it to run and operate smoothly. It also contains all the game-loop functions. Herein the game also gets initialized and started up. It contains a wide variety of operations. Each section will be briefly explained, and important operations will be explained as well. A signature table will not be kept as it's not an ADT.

Level & Round Logic

- For the level & round logic, a giant function named round-logic has been made that takes input a round (from the round ADT) and a bird. The round is mapped over all existing birds, once a bird has spawned the birdamount inside the round is decremented, and based on the birdamount a next round is initiated. For the levels, a global variable is kept of the total amount of all levels combined, and this is used to progress the levels.
- The resetter operation resets the game.

Screenlogic

At the screenlogic section simple operations are kept show & deleting different screens at different intervals of the game.

Spawn Objects
The spawn objects section contains all of the operations that bring something into existence in the game. Take for example the (spawn-duck) operation. It uses the Duck ADT to make a duck, and adds it to a global duck list, then it uses the Draw ADT to draw it. All other "spawn" operations make an object with their respective ADT and add it to their respective global list.

Draw
In the draw section of the Game ADT operations are kept that combine the game & draw logic. These should have been incorporated into the Draw ADT but weren't due to their complexity and the fact they combine the front and back end.

Ammo Logic
The ammo logic section contains some complex code for a relatively simple feature. The reason for this complex code is the fact that when clicked, a shot is registered multiple times, therefore states have been implemented along with a time function inside of the functions that control the ammo to negate the effect of decrementing the ammo by too much.

Lives & Score Update
Here you can see two similar procedures, each being used to update the score & lives visually on screen for the player to see.

Kill & Remove Game Elements
This section is the opposite of the spawn objects section. It contains all the logic of when and how an object should be removed. In general, removing an object means taking it out of the global lists and removing their tiles.

Powerup Logic
As the name implies all of the powerup logic is contained in this section. Through a combination of procedures, it detects when a powerup has been shot, sets it to activated, does the desired effects, and when the timer ends undoes the effects and returns the game to normal.

Extra
Extra contains mainly timer functions, which fix the bug of a shot being registered multiple times.

Birdlogic
The all-birds procedure uses the internal operations of the duck adt to control all of the birds features inside the game.

Obstacles
Two operations are inside this section, utilizing the Hitbox ADT, obstacles are detected, and their desired effects are implemented.

Gameloop Function
This contains all of the above procedures & operations and implements them into the gameloop. There's a (spawn-bird) procedure at the beginning to kick off the gameloop.

In conclusion, the "Game ADT" file is a result of quickly adding code on top of one another without much thought towards organization or efficiency.

2.1.2 Position ADT

Since we are working in a 2D x-y grid a position ADT can be extremely useful. Its purpose is to make the operations on the grid easier and control all the movement on the back-end.

Name	Signature
make-position	*number, number → Position*
x	*Position → number*
y	*Position → number*
x!	*Position, number → Ø*
y!	*Position, number → Ø*
in-range	*Position, number → Boolean*
position!	*Position, Position → Ø*
distance	*Position → number*

Because the Position ADT is such a crucial element, multiple operations besides the standard x and y values have been included to talk about the position.

- The make-position operation allows for the creation of a Position value. Given an input of two numbers it creates an (x y) pair of coordinates.
- Both x and y performed on a Position give their respective x and y values
- Both x! and y! allow for the modification of respective x and y values by new values when performed on a Position.
- The in-range? operation checks if two Position values are in the range of each other within a given radius, it does this by calculating the distance.
- The position! operation takes a Position, and updates it with a new Position, thus "moving" the original Position.
- The distance operation takes two Position values and gives their distance.

2.1.3 Draw ADT

The Draw ADT contains all the front-end logic & game-loop operations, it also initializes the game and brings certain things into existence (ex: define gun). It is actually not an ADT, but just a file containing a lot of different operations. See explanation in intro.

Name	Signature
display-tile-for-duck	*Ø → Ø*
display-tile-for-bomb	*Ø → Ø*
display-tile-for-flight	*Ø → Ø*
display-tile-for-chicken	*Ø → Ø*
display-tile-for-pigeon	*Ø → Ø*
display-tile-for-crow	*Ø → Ø*
display-tile-for-droplet	*Ø → Ø*
display-tile-for-focus	*Ø → Ø*
display-tile-for-chaos	*Ø → Ø*
draw-droplet	*duck → Ø*
find-tile	*object, tag → tile*
remove-tile	*object, tag → Ø*
make-invisible-duck	*duck → Ø*
make-visible-duck	*duck → Ø*

- All the “display-tile-for” operations take for the given object their respective image, add it to the layer, and add the tile to a global list that’s used to manipulate the tiles.
- The draw-droplet operation is needed for the water logic of the game.
- The find-tile operation gives an object, its respective tile. This makes it a lot easier to combine game logic and draw logic.
- The remove-tile operation removes a tile by removing the drawable and removing it out of the list.
- The last two operations are used to give the appearance of a duck behind and a duck in front of an object.

2.2 In-Game Objects

2.2.1 Duck ADT

The Duck ADT describes the needed behavior of all the birds inside the game.

Name	**Signature**
make-duck	*Position, tag → Duck*
position	*Duck → Position*
tag	*Duck → Symbol*
wall-touching?	*Duck → Boolean*
wall-counter	*Duck → Number*
angle	*Duck → Number*
alive-status	*Duck → Boolean*
obstacle-status	*Duck → Boolean*
animation-blocker	*Duck → Boolean*
lives	*Duck → Number*
pigeon-timer	*Duck → Number*
allow-timer	*Duck → Boolean*
pigeon-on-ground	*Duck → Boolean*
pigeon-open-timer	*Duck → Boolean*
obstacle-randomizer	*Duck → Number*
score-stopper	*Duck → Boolean*
watered	*Duck → Boolean*
caught-in-net?	*Duck → Boolean*
removed	*Duck → Boolean*
dead!	*Ø → Ø*
alive!	*Ø → Ø*
inside!	*Ø → Ø*
outside!	*Ø → Ø*
animation-blocker-false!	*Ø → Ø*
animation-blocker-true!	*Ø → Ø*
minus-life!	*Ø → Ø*
score-stopper-false!	*Ø → Ø*
water-yes!	*Ø → Ø*
water-no!	*Ø → Ø*
reset-crow	*Ø → Ø*
crow-logic	*Ø → Ø*
duck-movement	*Ø → Ø*

make-slow	Ø → Ø
net-yes	Ø → Ø
net-no	Ø → Ø
removed!	Ø → Ø
pigeon-logic	Ø → Ø
reset-pigeon	Ø → Ø
move!	*number* → Ø
random-angle	Ø → Ø
make-slow	Ø → Ø
walls	Ø → Ø
change-angle	Ø → Ø

- The make-duck operation creates a duck with a position and a tag, the tag is used to identify which type of bird is being created here.
- The duck gets initialized with some information; this information contains.
    - Touching a wall or not → wall-touching?
    - Current wall it is on → current-wall.
    - Number of times it has hit a wall → wall-counter.
    - An initial angle → angle
    - Information for different states & guns is also defined, as well as the behavior depending on the tag.
- Position gives the birds position.
- The move! operation moves the bird with an input that'll be used as the angle. Using cos and sin and a step amount (time), a new position is calculated along the angle. Move also contains the unique movement for when a duck has died and unique movements for the pigeon, crow & chicken.
- The duck-movement contains the actual logic of when and how a bird should move, it's exported for use inside the gameloop.
- Pigeon-logic & crow-logic both use timers to reset the birds' respective unique movements. Since these movements (sitting still and hiding inside an object) both contain one-way Booleans that turn false after being completed. Timers are used to reset these Booleans back and give the birds another chance to perform their unique movement. Reset pigeon – and crow is a simple procedure that changes the actual Booleans.
- The make-slow procedure along with the other "water" operations are used to communicate with the water gun on when to slow a bird down etc.… The exact same thing holds true for "caught-in-net?" and other "net" operations, but this time for the net gun.
- The set! Procedures that are inside! outside! dead! And alive changes the bird from inside on obstacle to outside, from dead to alive. Obstacle-status & alive-status are used to call upon the status of the bird.
- The random-angle operation looks at which wall the duck is on and then gives it a random angle based on that.
- The walls operation contains all of the logic, it constantly updates wall-touching, wall-counter & current-wall.
- Change-angle changes the duck's angle to a random one.

Overall, the Duck ADT contains a lot of information about the different birds that needs to be broadcast, as well as the logic of different birds.

2.2.2 Bomb ADT

The Bomb ADT describes the needed behavior of the false target inside the game.

Name	**Signature**
make-bomb	*Position* → *FalseTarget*
position	*Bomb* → *Position*
move!	*Ø* → *Ø*
set-spawn-point!	*Number* → *Ø*
set-left-zero!	*Number* → *Ø*
top	*Bomb* → *Number*
alive-status	*Bomb* → *Boolean*
animation-blocker	*Bomb* → *Boolean*
animation-blocker-false	*Ø* → *Ø*
animation-blocker-true	*Ø* → *Ø*
sidestep	*Bomb* → *Number*
width	*Bomb* → *Number*
dead!	*Ø* → *Ø*
random-top	*Ø* → *Ø*
random-sidestep	*Ø* → *Ø*
random-width	*Ø* → *Ø*

- The make-bomb operation takes as input a position type and makes a false target.
- The bomb is initialized with values relating to the parabola function such as the top, sidestep, width, the spawn-point & left-zero. These can be called up with their respective functions.
- The position operation gives the position of the false target.
- The move! operation moves the false target using a parabola formula, it calculates the next position through the step-size.
- The "random" operations all change the respective initialized values to a random one, this is used to randomize spawning of the bombs.
- Set-spawn-point and set-left-zero both take as input a number and set it to the bombs spawn-point and left-zero.
- The animation-blocker operations are used for the dead animation of the bomb, to not have the Boolean constantly switching back and forth a Boolean is used to close a branch of the animation gate inside of the Game ADT. The alive-status and "dead!" operation also communicate to the outside of when the false target should be put as dead, and also making it die.

2.2.3 Droplet ADT

A droplet is needed & made to keep track of a unique position. When a bird has been hit with the water effect, the droplets position keeps track of the bird's position and this way a status

effect can be drawn.

Name	Signature
make-droplet	*Position → Droplet*
position	*Droplet → Position*

- The make-droplet operation takes as input a position type and makes a droplet.
- Asking for a position gives you the droplets position.

2.2.4 Powerup ADT

The False Target ADT describes the needed behavior of the false target inside the game.

Name	Signature
make-powerup	*Position, Tag → Powerup*
position	*Bomb → Position*
timer-increment	*Ø → Ø*
stop-timer!	*Ø → Ø*
start-timer!	*Ø → Ø*
timer	*Powerup → Boolean*
powerup-timer	*Powerup → Number*
activate!	*Ø → Ø*
deactivate!	*Ø → Ø*
active	*Powerup → Boolean*
tag	*Powerup → Symbol*
reset-timer!	*Ø → Ø*

- The make-powerup operation takes as input a position type and a tag and makes a powerup.
- A powerup has an active time, therefore functions timer-increment, stop and start-timer, reset-timer are implemented to work on powerup-timer.
- A powerup has a tag to identify it, either chaos or focus and can be called with the tag dispatch.
- A powerup has an "active" state that can be activated or deactivated, so the powerup can communicate with the game loop whether to perform its functions or not.

2.3 User-controlled

2.3.1 Gun ADT

The Gun ADT is another crucial part of the game, most dynamic code will depend on the gun's interaction with the game. The graphics library will also be incorporated into the gun's functions.

Name	Signature

make-gun	*Position → Gun*
position	*Gun → Position*
shot?	*Gun → Boolean*

- The make-gun operation takes as input a position type and makes a gun.
- The gun is initialized with a shot? value set to #f.
- The position operation gives the current position of the gun, using the mouse-move-callback t will constantly update the position as the mouse moves.
- Using the mouse-click-callback function the shot? Boolean can get changed from true to false. The shot? operation gives a Boolean representing if a shot has been taken or not.

2.4 Stats

2.4.1 Score ADT

The Score ADT will keep track of the users score.

Name	**Signature**
player-score	*number → Score*
score	*Score → number*
plus-score	*Ø → Ø*
plus-score-chicken	*Ø → Ø*
plus-score-crow	*Ø → Ø*
plus-score-pigeon	*Ø → Ø*
plus-score-extra	*Ø → Ø*
reset-score!	*Ø → Ø*

- The player-score operation takes as input a number, an initial score and makes a score.
- The score operation gives the current score.
- The plus-score adds 20 points to the score, the minus-score isn't currently used in the game but is useful to have. There are also respective plus-score operations for the birds, that each add their own amount. Plus-score-extra is a special amount for when a bird is caught with a net.
- A reset-score operation is provided here to reset the game.

2.4.2 Lives ADT

The Lives ADT will keep track of the users' lives.

Name	**Signature**
player-life	*number → Score*
lives	*Ø → number*
plus-life	*Ø → Ø*

minus-life	Ø → Ø

- The player-life operation takes as input a number, an initial score and makes lives.
- The lives operation gives the current score.
- The plus-life adds 1 point to the score, the minus-life isn't currently used in the game but is useful to have.

2.4.3 Round ADT

The Round ADT contains all the logic needed for a round & level.

Name	**Signature**
make-round	*number, number, symbol, number, number, number* → *Round*
level	*Round* → *number*
round number	*Round* → *number*
birdamount	*Round* → *number*
roundstart	*Round* → *Boolean*
round-start-no!	Ø → Ø
birdamount-decrement!	Ø → Ø
bombamount	*Round* → *number*
bombamount-decrement	Ø → Ø
type	*Round* → *symbol*
extrabirdamount	*Round* → *number*
double-birds	Ø → Ø
reset-rounds	Ø → Ø
powerupamount	*Round* → *number*
round-decrement!	Ø → Ø
powerup-decrement!	Ø → Ø

- The make-round operation takes as input the following information, the level, the round number, the type of round (normal hard or extreme), the bird amount, the bomb amount and the powerup amount.
- There are decrementer functions for the bomb, bird, and powerup amount to each update when one of them has been killed.
- The double-birds operation is used for the chaos powerup. Extrabirdamount also exists for this reason to keep track of the original bird amount to change it back when the chaos powerup effect wears off.

2.5 Extra

2.5.1 Hitbox ADT

The Hitbox ADT is primarily used in obstacle logic, it receives 4 coordinates to make an

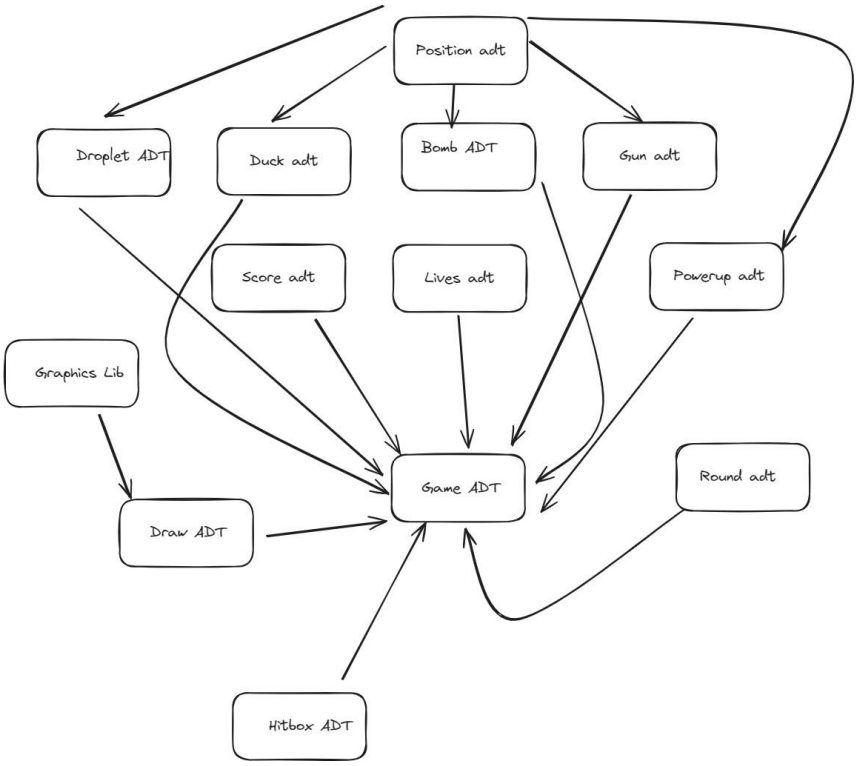
active hitbox.

Name	Signature
make-hitbox	*number, number, number, number → Hitbox*
inside?	*number, number → Boolean*

- The make-hitbox operation takes as input four coordinates and makes a hitbox.
- The inside? Function takes as input 2 coordinates (x, y) and returns a Boolean that is true or false depending on if the coordinate is inside the hitbox

3 Dependency Diagram

The following figure displays the implementation's dependency diagram.



Since the Game ADT contains the bulk of the code, it is dependent on all the other ADTS. You can also see that the position ADT is a very crucial element to the game as it's used in every single object. Other than that, ADTs don't really communicate with each other or rely on each other.

4 Schedule

The timetable of when the project was worked on wasn't kept but it was worked on every. weekend. Most of the implementations took as long as expected, but the length and complexity of the level & round implementation was underestimated. Overall, the schedule was maintained.