



V8078659

Francesca Porritt

GAV2016-N - Games Engine Construction

CHICKENS 2 Demo

User Guide:

How to play the demo:

The player must protect the Chickens from Cats while making sure they are well fed so they will lay their eggs. The player must collect as many eggs as they can within the time limit!

To feed the chickens, the player must go to the seed box and collect the seeds.



When the player has seeds they have to put them in the chicken feeder for the chickens to eat.







After eating the seeds the chickens will lay their eggs in the coop for the player to collect.



If the player doesn't scare away the cats they will eat the chickens, to scare away a cat the player must throw a rock at them.



Control Guide:

	Move		Throw
	Use/Interact		Pause

Known Issues / Bugs:

- The refill timer above the seed box doesn't clip properly.
- If the environment entities are included in the collision loop it creates an invisible wall halfway through the play area, so they have to be skipped in the loop.
- When quitting via quit option in menu, the deconstructor in world doesn't get called so is causing memory leaks. I've attempted to call the deconstructor from there and also putting the delete calls in the quit function but that just causes errors.
- When the cat is hit by a rock, it plays a sound. The call to enter the leavingState however gets called when the rock is colliding which happens over multiple loops meaning the call is made multiple times and thus the sound is played more than once as the cat leaves.
- When the game is paused the clock continues meaning when it's unpaused the timer skips ahead and the player loses time.

Implementation Checklist:

Implementation check list

Adequate Rated Items (D to C)

Graphics

- ☒ A 'black boxed' graphic system is in place
- ☒ Textures can be efficiently drawn to arbitrary positions on and partially off the screen (clipped)
- ☐ Animation is implemented and working correctly

World State

- ☒ A player entity exists
- ☒ Input is recognised and can be used to alter the world state e.g. move the player entity
- ☒ The Xbox controller is supported and 'hot pluggable'

Code Quality

- ☒ Class interfaces are minimal and complete. Class function and member variable visibility is correct
- ☒ Code can be built and executed without compiler errors or warnings in debug and release
- ☒ Code is well commented
- ☐ There are no memory leaks *→ ONLY WHEN QUITTING THROUGH MEMORY*
- ☒ There is good error handling throughout
- ☒ You have followed all the submission requirements e.g. made a video, submitted the correct files etc.

Report

- ☒ All requested sections have been attempted adequately and the report is professionally presented

Good Rated Items (C to B)

World State

- ☒ A world model system is in place. It is separate from other code and black boxed
- ☒ There is a game loop handling input, world update and rendering
- ☒ Bounding rectangle collisions are detected
- ☒ There are multiple world entity types

Code Quality

- ☒ Good use of object-oriented techniques e.g. polymorphism, member variable visibility
- ☒ Memory is only allocated / deallocated outside of the game loop
- ☒ Const is used correctly

AI

- ☒ Some AI routines are in place e.g. enemy entities move around the world following paths, use state machines etc.

Report

- ☒ This report would allow another programmer to work with your code systems

Other

- ☒ Some sound effects are in place

Excellent Rated Items (B to A)

Graphics

- ☐ Interpolation is used to smooth entity movement

World State

- ☒ The player entity can shoot projectiles (or equivalent functionality) *ROCKS*
- ☒ Explosion and bullet management *→ ONLY ONE BOMB STORED BUT CANS USED MULT TIMES SH. TO BULLET*
- ☒ Game play is independent of platform capabilities (i.e. uses a model tick approach)
- ☒ Game cycling e.g. detection of win / lose conditions and restarting the game
- ☒ There is a scoring system with the score shown on screen

AI

- ☒ Several different enemies with differing behaviours

Report

- ☒ This report has insightful and balanced reflection

Extra Marks (Examples)

- ☐ Mapping of world space on to screen space
- ☐ Other graphics techniques have been implemented e.g. background scrolling, blending modes etc.
- ☐ Level data is loaded from a file
- ☐ A difficulty level
- ☒ More advanced C++ e.g. use of namespaces, STL, C++ 11 and patterns
- ☒ 'Intelligent' enemy behaviour *MAYBE!*
- ☒ There are sound effects for collisions, explosions and firing
- ☒ Additional black box systems have been implemented e.g. for AI, Sound
- ☒ HUD features beyond simple text e.g. health bars, mini maps etc.
- ☐ Other features, please list below:
FINITE STATE MACHINE

Maintenance Guide:

Visualisation:

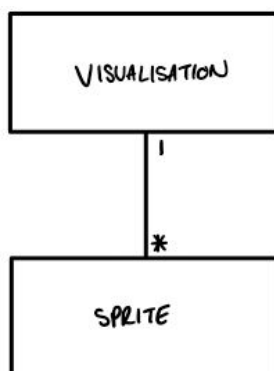
The visualisation system handles all the rendering within the program and sprite handling. Nothing outside of the program knows anything about drawing, and visualisation knows nothing outside of itself.

This system only has one function linked to my game demo, the `timerClipBlit`, making it easy to reuse the system in another game. Even though the timer function was made specifically for my demo it could easily be reused for another game as a health bar, timer, etc.

The advantages to not having a strong link to my demo, means that it can easily be reused for future games and enable rapid prototyping.

Classes in the system:

- Visualisation
- Sprite



World:

The world system handles the bulk of running the game and controls the state of the game based on user input and events.

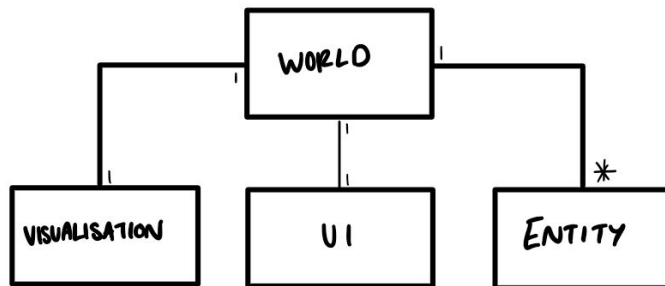
Currently the system is fairly linked to my game demo. However, while no code within visualisation would have to change to work in another game, by simply removing code in the key functions it could easily be reused in another game. For example, the `run` function calls `update` which calls the functions based on the current state, ie. if `state == play`, call `Play()`. This would work for any game but the contents of the `play` function would have to change to suit the new game. There are also four extra functions unique to my game but our only called with `play`, so as the code in `play` is removed these functions can be removed as well.

The advantage to this link is that it works well for my demo and is easily customisable, I would actually add a few more unique functions to the class to make it smoother and neater for my demo rather than having all the code within `Play()`. The disadvantage to it is simply the time and effort it would require to reuse, while I am confident it will work in another game

after being cleared out there could be some artefacts left over that would require some rewriting.

Classes in the system:

- World
- Visualisation
- Entity
- UserInterface



Conclusions:

Final implementation:

Overall, I'm proud of my final implementation. There is a lot I would like to add and I would like to clean up the code a bit as the play function within world can be quite overwhelming at times. For example all the UI calls could be moved into a function within world and would then just need one call in Play(), but I think I came up with a unique game demo idea and executed it fairly well.

Lessons learned:

- I really enjoyed learning about finite state machines and being able to implement two into my project.
- For this project I started keeping a journal in a notes.cpp within the project. Initially it was meant to just be a place to note down what I was currently working on before stopping so that when I came back to it I could remember where I was at, but it evolved into a way for me to plan out what I was doing and make notes of what worked, what didn't and how I fixed things. It's something that I've found extremely beneficial to have to look back on and will take with me to future projects.

Future enhancements:

- In the future I would add a second enemy type, such as a bear, that is faster and takes multiple hits to scare off.

- Currently the AI types have a set speed function so their speed can be changed during run time. I would like to add this to the entity base class so I could introduce pick up items that affect the players speed and such.
- I would create a new AI class that inherits from the Entity class with the Chicken and Cat classes as children. I had already implemented the Chicken class when I began the Cat class and then realised it had a lot of the same variables as the Chicken class. It would have definitely have been better to have those common variables and functions within an AI class.
- The Chicken entities would also have been better suited to being stored in a list rather than a vector within the World class. Currently when the cat eats a chicken, the chicken is just set to inactive but had they been stored in a list it would have been easy to completely remove it from the container instead.
- I really wanted to add animation and after creating the player, it was one of the first things I started working on. Unfortunately, I struggled to get it to work and decided to save it to the end so I could focus on the rest of the demo but have run out of time.
- I also had issues to get the restart after the demo finishes to work, so this will be fixed in the future.
- I had also intended on adding difficulty levels, where chickens/cats increase/decrease depending on the level of difficulty.

Tested on TU46937@WINDOWS.TEES.AC.UK

Assets used:

Stardew Valley Sprites by [Ninth World](#)

[Sounds](#)

[04b_30 Font](#)