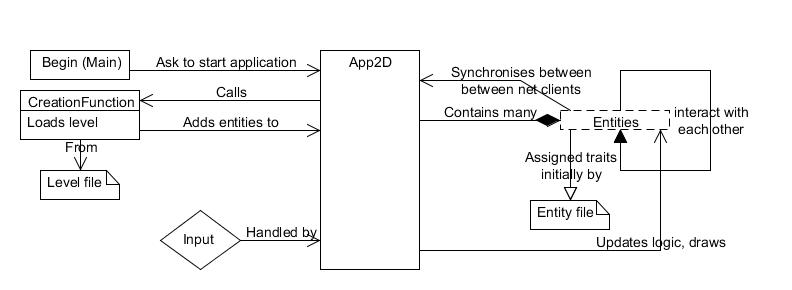
**Engine Architecture** – Pre-Modification

*Seb Holzapfel*

This is a (largely informal) description of the accelerated 2D application engine I developed to provide for the needs of 2D projects (games or productivity), nicked **App2D**.

It has evolved to include more features and be more efficient as I use it in more and more projects (specifically, I have used it in 3 projects; a picture frame application, a 2D physics simulator and a multiplayer (extremely customizable) arcade space-shooter called Fluxan). I did not develop the framework with a specific development methodology in mind, however looking back the practices I followed are most likely related to the Agile approach as requirements changed quickly and iterations were fast.

To understand how the framework operates, an overly simplified visual representation may help:

As is visible, App2D basically manages all application entities, assigning them traits according to a file description. It also handles input, passing it to all entities which require input (such as the player entity). Each entity trait (referred to as Tags) has 4 functions, Init, Step, Draw and Destroy, these are automatically called when an object is created, every frame, when the world is being drawn, and when the entity is destroyed; respectively.

This approach has flexibility as an entity is not simply a large pre-defined object, it is loaded from various different entity ‘components’ at runtime. For example, here is the entity definition file for a ‘Death Warper’, used in Fluxan (dwarper.edef):

NAME dwarper

FILE dwarper

SCALE 2.5

ROTATES -20

WOBBLES 0.05 2

CREATES\_GRAVITY 600 50 1

USE\_RANDOM\_ROTATION

DESTROY\_TYPE\_ON\_RADIUS b 10

DESTROY\_TYPE\_ON\_RADIUS scrap 30

DESTROY\_TYPE\_ON\_RADIUS remoteplayer 30

The exact semantics of each instruction is described in the ‘Tag list, pre-modification’ document, but a brief outline follows:

The first line is the name that the game will use to reference this entity, the FILE is what the object’s shape file is called (Fluxan used line-graphics, shapefiles are just lists of lines with colours. For this project, I may need to use images and have to change this system). The SCALE element adds code to the entities’ initialisation routine, scaling it by 2.5. The ROTATES element adds code to the entities’ Step routine, making it rotate 20 units per unit of game time to the left every frame, and so on.

Each ‘Tag’ is represented by a class which is added to the entity at runtime if it appears in a file. All entities can access App2D’s entity repository and so can check for collisions with each other, enforce gravity etc. Each entity keeps a reference to its parent App2D class (There should be only one).

App2D supports TCP/IP multiplayer with an effectively infinite number of players. This feature will be removed as it is not required in my project, however I will outline how it works anyway. On an application being executed, the user must decide whether the application will operate in server or client mode. If it is a client, it does not have permission to create or remove entities (unless it does not receive an entity update packet within a specific timeout, in which case the entity will be deleted as most likely a deletion packet was lost). If it is a server, there is no player entity – only the game logic is executed and all players are treated as remote players. The server regularly broadcasts updates for all entities about 15 times a second (Not at the application frame rate as that would clog the TCP pipe). This includes entities’ velocity so that clients can predict what direction entities will move in and reduce jerkiness. Whenever an entity is created or destroyed on the server, a special creation or deletion packet is sent to all clients to perform the same.

The only information sent from the client to the server is the movement of that player’s ship, requests to create projectiles emanating from that player, and chat messages.

Implementation was of course much more complex than this, but the principles are as described.

App2D uses 2 other 3rd party libraries, SFML and Boost. SFML (Simple and Fast Multimedia Library) is used for accelerated 2D rendering, audio and some TCP routines. Boost (A group of high-performance C++ utility libraries) was used for file system traversal and a special type of pointer that keeps reference counts to itself (boost::shared\_ptr). App2D was previously also integrated with Box2D (A 2D Newtonian physics simulation library) for a small physics demo I wrote, but dependency on this library was removed in my latest project as it wasn’t needed.

Just for the sake of it, here are a couple of problems (There are of course more) which I distinctly remember running into when I was developing the framework:

* When entities deleted one another, sometimes the application would try and update the entity that was deleted (in the same frame) even though it didn’t exist (Access violation!). Now, when an entity is deleted it is not actually deleted until all entities have been updated (it is added to a queue which is executed at the end of every frame)
* The user interface and the game objects operate on a different coordinate system, as the user interface is always stationary, and the game camera is moving around. I ended up using 2 draw ‘phases’ where the camera transform changed between the two.