cgame: A rapid game development tool

by Nikhilesh Sigatapu, advised by Prof. Adam Finkelstein

Princeton University, Department of Computer Science

Introduction

cgame is a 2d game engine and editor for **rapid game development**. Its design was driven by four key concepts:

- Entity-system architecture
- Persistence of data
- Real-time logic editing (scripting)
- Real-time data editing (in-game editor)

Persistence

Entity data in cgame can be saved to a buffer (in-memory or on disk) and loaded back. Saved buffers can be ***merged*** into the current world — loading saved entities without destroying existing ones. While saving, it is possible to ***filter*** to save selected entities only. Saving and loading, especially with merging and filtering mixed in, allow for various useful functionality:

- Duplication (filter to entities, save to memory buffer, merge back)
- Prefabs (save entities to files, then merge whenever needed to instantiate)
- Undo/redo in editor
- Save player progress through the game

Scripting

cgame's core systems and data structures are mostly written in C, but are also exposed to Lua, an **interpreted scripting language**. Systems can be written in Lua too.

This allows **interactive programming** like at a Python prompt. Scripts can be written, interpreted, modified and interpreted again at run-time. It is possible to create an entirely new monster type, add it to current scene, test combat and fix bugs, all without exiting the game.

Lua systems can have their data be **automatically saved and loaded**, and their properties are **automatically discovered** by the in-game editor.

Entity-System Architecture

Entities are dynamic objects in a game such as the player, a monster, a camera or perhaps even a GUI button. The entity-system model **splits entity data and functionality across *systems*** like in a relational database. This is in contrast to the object-centric model: a single structure for each object with all data and functionality. Advantages:

- Dynamically mix-and-match entity features
- Cache-friendly
- Save/load-friendly

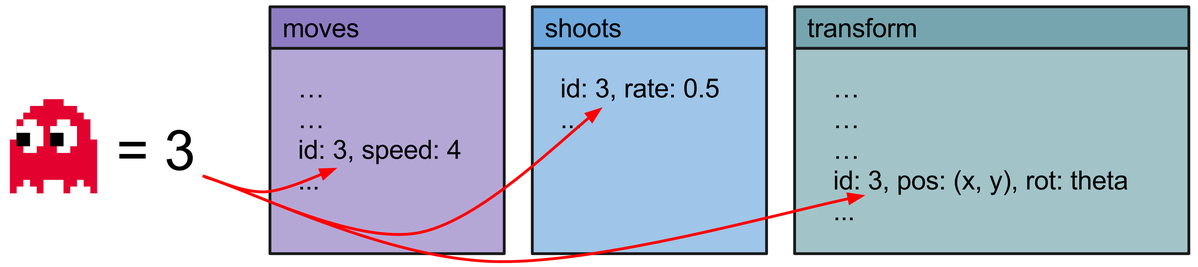

Figure 1: Mixing `moves` and `shoots` systems to make a moving, shooting monster

In-Game Editor

cgame provides an in-game editor that allows the user to select entities, move and rotate them visually, edit properties and add and remove entities from systems. Game logic can continue to run while in edit mode. **Property changes are reflected in real-time**, so, for example, decreasing the `speed` property of the bullet below would make it immediately slow down. The editor automatically discovers properties and data types for script systems. Script systems added at run-time are available to be inspected as soon as they are defined. The editor is written almost entirely in script and allows the user to create custom extensions.

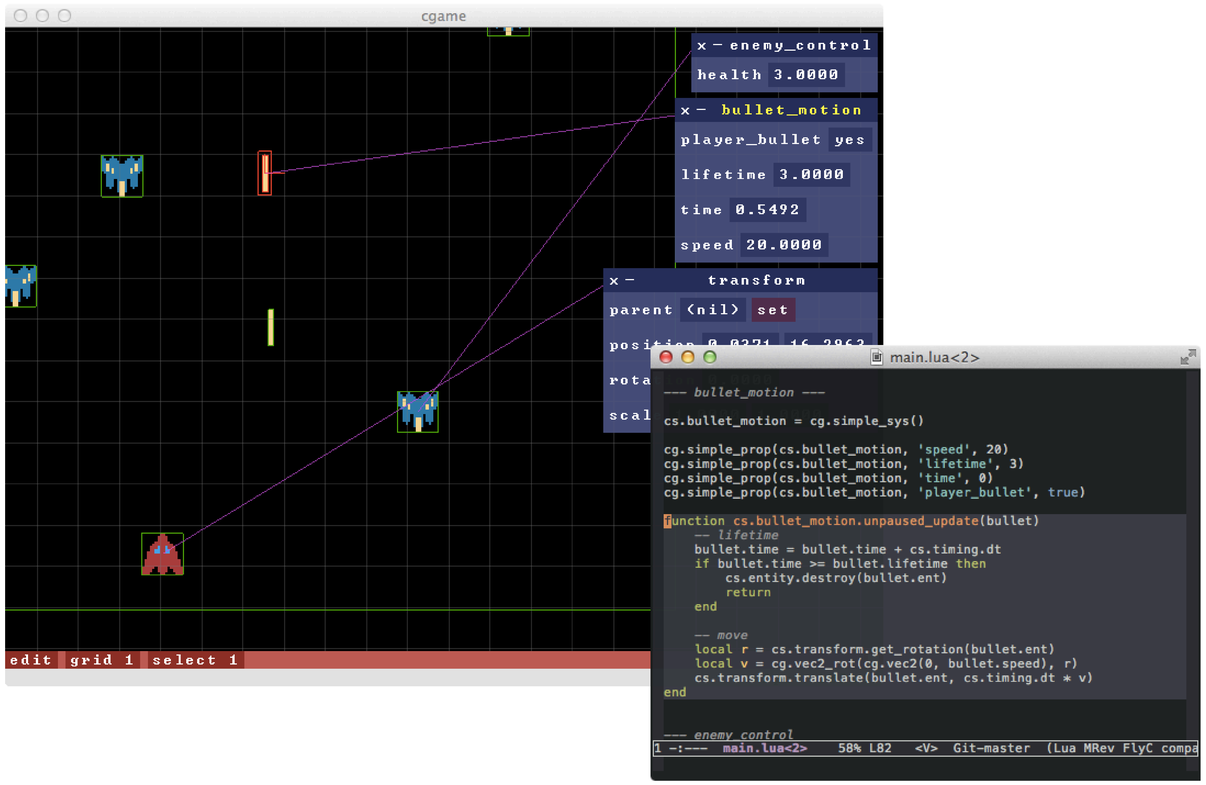

Figure 2: Live-coding a space shooter game while editing a paused game level

One Hour, One Prototype

A time-lapse video of cgame being used to develop the space shooter game prototype pictured can be found at `https://www.youtube.com/watch?v=Lb1NWyUqAfA`. The game was developed completely from scratch in one hour, including drawing images.

Limitations and Future Work

- Inter-system **dependency management**. The `sprite` system depends on `transform`. What should happen if one of its entities is removed from `transform`?
- System-specific **event notification**. Currently events are notified through polling (checking every update), but what if the listening system is updated before the event value is set? Ties back to inter-system dependency management.
- Save/load **version management**. Currently save buffers become unusable if new fields are added or old ones removed in C systems.

These issues point to the need for a better way to manage **system metadata**, including dependencies, events subscribed to and properties that must be saved and loaded. Save/load could be made backwards compatible by having the metadata itself be saved in the buffer.