turnt.

Khettai Laksanakorn (`kl2679`)
Noah Stebbins (`nes2137`)
Jeff Tao (`jat2164`)
Whitney Bailey (`wvb2103`)
Xiwei Feng (`xf2140`)

24 February 2015

Contents

1 Introduction

The name of our language is “turnt”. This name is somewhat of a pun signifying the type of problem we seek to solve. Turnt seeks to solve a problem encountered by many programmers of all levels of expertise. Although this problem persists in Computer Science and Computer Programming today, many languages cease to make its solution a priority. Thus, the problem continues to be shoved to the wayside and pushed under the rug. The challenge I am speaking of is the writing of turn-based games. Any game in which a number of players take turns to influence game state is a turn-based game. Games such as spades, monopoly, and all sorts of card and board games fit into this category.

Although a number of programming languages have a methodology for writing games, these languages often fall short because it is not their main goal. The main goals of popular programming languages today seem to be more geared toward broader goals in Computer Science theory, and providing a way to write games is just an after-thought in the mind of the creator if the thought occurs at all. Our language seeks to facilitate the writing of turn-based games by having an event-based ideology. This event-based ideology is explained below in greater detail. Our language compiles to Java and is suitable mainly for individuals who have some programming experience. You do not need to be an expert in programming to use our language but those who have basic skills in programming are our target audience. If you have ever tried to make a relevant game for you or your friends to use for studying or entertainment, please keep reading because this is the language for you.

2 Background

2.1 Turn-Based Games

Turn-based games encompass all games in which one or more players perform (“take”) actions (“moves”) in a pre-defined series of steps (“turns”). A turn is the atomic temporal unit in a turn-based game. It typically can last an indefinite amount of time, and the key feature of a turn is typically (but not universally) that only one particular subset of players (often only one individual player) may make moves during their turn. The actions available for a player to take are defined by the rules of the game, which are typically set down textually and define the passage of turns, what moves are valid at what times, and win and loss conditions for the game. Rules are typically referred to in disputes of legal play.

Turn-based games fall under a variety of genres. Board games (e.g. chess) involve a physical field of play (“board”) often with discrete locations (“spaces” or “tiles”) upon which movable objects (“pieces”) are arranged. Card games utilize decks of playing cards as pieces. Card games may also involve a board (e.g. solitaire). Many card games are based on the "standard 52-card deck". Still, others (especially so-called "trading card games") consist of a wide variety of playable cards, which may optionally include additional rules for their play. The "rule text" of a played card can modify the game currently being played by adding extra conditions to moves made following the introduction of the card in question, and this rule text takes precedence over existing rules, including other rule-modifying cards.

2.2 Event-Driven Programming

The event-driven paradigm often sees use in graphical and web applications where the event model allows for the programmer to define asynchronous responses to various user input events (e.g. mouse clicks, keyboard presses). This model has recently gained traction due to the rise in popularity of Javascript-based web applications.

Event-driven programming as a paradigm presents an alternative to imperative programming that lends itself well to turn-based games with well-defined rules. Rather than forcing the programmer to keep track at all times of what actions need to be taken to handle a particular player move, the event model allows handling functions ("handlers") to "listen" to an event. When an event is emitted, all listening handler functions are called in succession. This model translates particularly well to the rules of a game, which are often phrased in terms of what action should be taken when a particular move is made.

The system of event handlers even allows rules to be mutable. If a particular handler is the "standard" way to modify game state in reaction to a move, then it is possible for moves taken to fundamentally modify the way that the game is played by registering new event listeners. Indeed, there are even games fundamentally based around this mechanic of mutable rules, so-called "Nomic" games. The mutability of rules also vastly simplifies the introduction of extremely specific rules, such as those found in trading card games, whose rules are often mutated according to the rule text of cards currently in play.

3 Related Work

- JavaScript events
- HDL (VHDL)
- Visual Studio

With the rapid development of the game industry, programming languages capable of game design have appeared. Early games were normally designed in process-oriented programming languages like C. Although very powerful, these languages are difficult to learn and not user-friendly. C# is an object-oriented language introduced by Microsoft for developing applications with a graphical interface and is one the most widely used languages in game design.

C++'s features help programmers with encapsulation and third-party libraries. However, for those who have little experience with object-oriented programming, *C#* is difficult to learn and requires the programmer to figure out many details in order to make the whole program run correctly. Another popular language for gaming is JavaScript. This language is event-driven with functions that are called on event triggers. Many popular online games are designed in this language. However, it does not provide a 1-to-1 translation between how you would write rules of a game and the programming language itself.

4 Goals of turnt.

Turnt is an intuitive, high-level, event-driven programming language designed to facilitate the coding of turn-based games using an event-action model.

Event-Driven

- As an event-driven programming language, the flow of a Turnt based program is determined by various events and actions. By providing a control flow more suited to turn-based games, Turnt streamlines the process of designing potentially complex game interactions.

Global State

- Turnt programs store the entirety of their state in a globally accessible structure. This simplifies the issue of scope, as many games feature complex interactions within the state.

Intuitive

- Turnt facilitates the implementation of a turn based game in a direct manner by allowing the programmer to work using constructs (rules and events) and a structure (actions and turns) corresponding to real-world game concepts. Because of this, Turnt is intended to be conceptually simple to use.

High Level

- By concealing the complexities of implementing an event-driven structure, Turnt creates a powerful abstraction for turn based games. Without the distractions of the code and design patterns underlying the event-driven turn structure, users of the language will be able to quickly and easily design a turn based game.

Portable

- By compiling a Turnt program into Java source code, Turnt output programs are compatible with any device capable of running a Java Virtual Machine. Additionally, the Java output produced by Turnt can be integrated with a larger Java program.