G!
A programming language for 2D games
Language Proposal

Rachit Parikh rnp2102@columbia.edu
Divya Arora da2254@columbia.edu
Steve Lianoglou sl2585@columbia.edu
Amortya Ray ar2566@columbia.edu

COMS W4115: Programming Languages and Translators
Department of Computer Science
Columbia University

September 26, 2006

Introduction

The minutia of game development is a tedious and complicated affair. In general, game developers are forced to write repetitive code that requires a lot of bookkeeping in order to ensure its proper function. If one takes a moment to think about the details involved in writing code to figure out if two objects run into each other, or moving an object on the screen in response to a keyboard press, the details of this process will quickly come to the light.

Enter **G!** The goal of G! is a game developing language which specializes in making interactive 2D games. G! will enable the game developer to focus on the overall game play of the target game instead of the rote details of the game play implementation.

G!

G! is a hybrid, simple, high-level, versatile, architecture neutral, portable, Newton-aware, rapid-development gaming language.

As the language designers who precede us have done, we have taken the liberty to characterize our language with a set of buzzwords. Each buzzword will be explained below along with the problems we are attempting to solve which resulted in their use.

Hybrid

Object Oriented Languages have been rising in language over the past several years. Some argue that this is a result of the expressive power that it lends to the programmer. Others would also argue that the previous argument is a fallacy and its true reason for popularity is to mitigate the impact of poorly written code to affect a larger system. We won't take a stand on this issue, per se, but, in general, we think OO-programming is *A Good Thing*™.

Sometimes OO programming can be too cumbersome and more traditional procedural programming techniques are *Good Enough*™ to solve a given problem.

G! will let you mix the two techniques in order to leave it to the developer to choose which approach is best suited for a given task.

Simple

What do we mean by simple?
The unique selling point of G! is that ANYBODY with minimal technical skills will smoothly be able to develop their own, unique gaming application. There will be a small number of keywords and constructs to remember with easy and intuitive syntax which will expose extensive libraries of functionality in an intuitive manner. This will simplify the process of writing game code.

High Level

The syntax and semantics of G! will allow the programmer to think of the game more on the lines of objects on a 2D plan as opposed to pixels in a Vector Space. The mechanics involved with writing code to move an image across the screen will be hidden underneath higher-level functions that will take care of these details for the developer.

Versatile

G! is a language which can be used to quickly and easily develop any type of game that can be played in 2D. Its functionality is not limited to writing only board games, card games, or the like.

If the game can be thought of as being played on a 2 dimensional plane, then G! is the right tool for the job.

Architecture Neutral

The game developer can program the game on the computer of her choosing. G! can be developed using Windows, Mac OS X, or Linux running on x86 or PowerPC architecture.

Portable

G! programs are compiled into java byte code, thus enabling the resulting programs to run on any machine that can run the Java Virtual Machine (Version 1.5 required)

Newton-aware

Part of developing games involves working with different bodies and how they react to each other. While Newton was focused on bodies of the celestial variety, G! deals with bodies that move around in the game. Game objects are collision-aware. No need for the developer to figure out how to determine if one thing touches the other, our game objects allow the developer to code up reactions when they are “touched”.

Rapid Development

Hybrid + High Level = Rapid Development!
In a few short lines of code you can have a rudimentary game going and a few short minutes is more than enough to come up with these few short lines of code.

Avenue for Productive Procrastination

G! makes you feel good about yourself. Are you slamming your head against the wall trying to build a linear classifier for data in n-dimensional hyperspace? Take a break and code up a quick game. You'll have all of your friends fooled as they think you're busily solving the problem at hand, when you're really having fun and making yourself a new game!

Synergistic

According to answers.com, 'Synergy' is defined as the interaction of two or more agents or forces so that their combined effect is greater than the sum of their individual effects."

We believe that co-operative interaction among the G! language code, the libraries and the compiler along with the JVM, creates an enhanced combined effect makes G! truly synergistic.

Basic Syntax

Here's what we envisioned our language to look like: The following snippet of code tries to define a rudimentary 2 player game. It basically does nothing more than play some sounds, do 'something' when one player crashes into another.

```
game = Game(name="Block World", num_players=2)
game.board = GameBoard(width=680px, height=680px, background_color=red)

#What is a game without some sort of a soundtrack???
game.add_sound("path/to/looping_sound_file.midi", key="bground_song")
game.add_sound("path/to/BANG_SOUND.midi", key="contact")

# Let's make a player object
# Load it from a picture -- the Sprite constructor will automatically
# Get from the image its size -- and make 'hit borders' to match

player1 = Sprite('path/to/player1_figure.png')

# So we have a sprite -- need to make it do things -- react to environment
# An Action library which defines basic actions
# like:
#    Action.contact
#    Action.move
#

my_action = Action.move(distance=5px, direction=Direction.left)

# what triggers actions? --> Events.
# There are EventHandler's which handle events that
# happen to/with sprites
handler = EventHandler()

# This handler knows how to react to key presses, in this case
# make the sprite move left 5px when left arrow (<-) is pressed
handler.on_key_press(key='<-', action=my_action)

# the handler also knows about events.
handler.add_behavior(trigger=Action.contact, action="Ouch!")

## now, make the player1 sprite react to these events.
player1.set_event_handler(handler)

# Repeat the same thing for Player 2
# ...
# ...


# What happens when players touch?

when (player1.touches(player2)):
    game.play_sound(key="contact") # play BANG defined above
    # triggers the behavior assigned to an Action.contact event on all internal
    # targets (here we say only on the player objects
    game.trigger_behavior(event=Action.contact, targets=game.players)
```

```
# ...
# ...

# on game startup
Sound.loop(game.get_sound(key="bground_song"))
```

The code above is meant only as an example to illustrate how we'd like the language to be lighter and quicker to write than Java. The syntax of the code borrows heavily from Python, however our intention is not to develop a Python clone (but we will take inspiration!).

Frequently Asked Questions

- How do you plan on doing all of this magic?
    - We have found a Java gaming library (http://goldenstudios.or.id/products/GTGE/GTGE) which we will utilize heavily. Our language will offer an elegant syntax for game development while leaving the heavy lifting of collisions and sprite movement to this library.

Conclusion

G! was one of those out-of-the-box ideas, that get conceived over a cup of coffee when four frustrated Computer Science majors, discussing computer games, sit wishing they could have a game that does this and a game that does that. And voila! They decide to come up with a language that allows them to do just that.

Like all things zen, things work most effectively when they're seemingly out of the way. The goal of G! is to lighten the burden of game development, allowing the programmer to create without struggle, do without trying, think without thinking.