

Teaching Computer Science through Game Design

Mark Overmars, Utrecht University

As a youth taking my first steps toward learning how to program a computer, I remember how exciting it was to write a program that printed the first 100 prime numbers. Nowadays, computer programs with fancy interfaces that provide access to music, video, and games have become commonplace, and such results no longer fascinate novice programmers. Instead, initiates want to be the ones who create these complex and attractive programs.

Unfortunately, even with the most wonderful application-development tools available, creating these programs requires a huge amount of work and a deep understanding of the computer system, programming language, available libraries, and development tool used.

How then do we transform computing consumers into creators?

LOGO AND LEGO

Logo (www.logosurvey.co.uk) and its many variants provide the classic example of a programming language aimed at creating interest among youngsters. Primarily seen as a language to make drawings, with Logo, the user steers a virtual turtle to draw shapes onscreen. Even the basic program can make fancy drawings this way, while modern versions extend Logo's possibilities considerably. For today's users, spoiled by console and



computer games, Logo is no longer flashy enough, however.

Steering a virtual turtle can't possibly compare with steering a real robot, which probably accounts for much of Lego MindStorms' success (www.legomindstorms.com). With the admittedly limited software that comes with this system, users can create and program their own robots. Fortunately, third-party developers have written complete programming languages for these robots, most notably NQC (<http://bricxcc.sourceforge.net/nqc/>).

The main disadvantages of using robots to learn programming are their expense and limited programming possibilities. On the other hand, robots do provide great vehicles for explaining concepts such as parallel tasks.

GAME DESIGN

Playing computer games is a popular recreational activity for young people. Not surprisingly, many of these enthusiasts dream that one day they will develop computer games themselves. So

why not use game design as a vehicle to teach youngsters computer science?

Developing computer games involves many aspects of computing, including computer graphics, artificial intelligence, human-computer interaction, security, distributed programming, simulation, and software engineering. Game development also brings into play aspects of the liberal arts, the social sciences, and psychology.

Creating a state-of-the-art commercial computer game is an incredibly difficult task that typically requires a multimillion-dollar budget and a development team that includes 40 or

Users who can't code a single line can use Game Maker to build games and learn object-oriented design.

more people. But simpler alternatives—ones within the reach of students and hobbyists—exist. Budding game developers can have fun creating variations on *Pac-Man*, *Space Invaders*, or simple platform games.

GAME MAKER

Writing a game like *Pac-Man* from scratch in a modern programming language is still difficult. Many gifted developers have embarked on such projects only to have their attempts end in frustration. Fortunately, several currently available tools make game creation easier.

With StageCast (www.stagecast.com), a tool that specifically targets children, the creator defines rules that link existing graphical situations to new situations. For example, a StageCast user can create a rule that states if an empty space lies to the right of a character, that character can move to occupy the space. Although intuitive, this approach can create only rather simple games. Likewise, ClickTeam

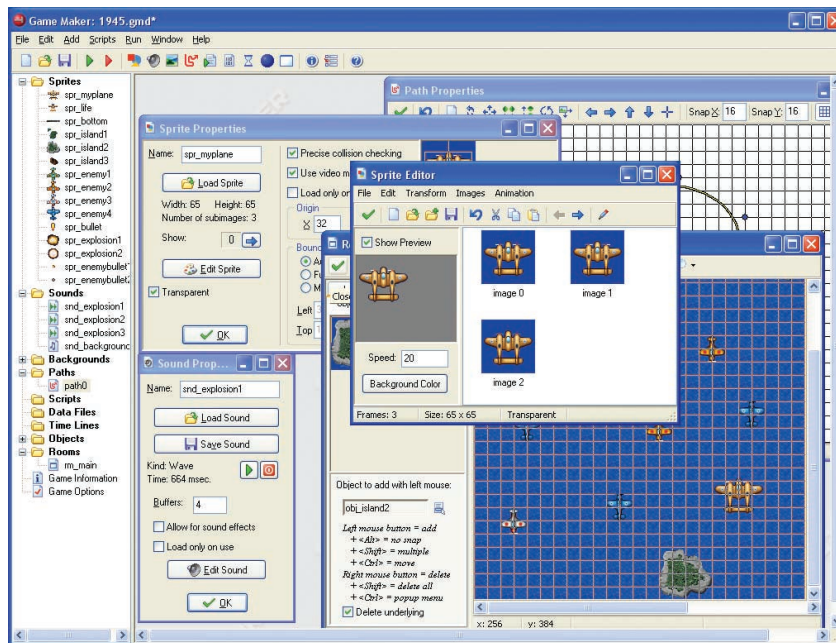


Figure 1. Game Maker interface. The left side displays resources such as sprites and sounds, while the right side shows the sprite editor, room editor, and other property forms.

(www.clickteam.com) produces several tools such as Click and Play and The Games Factory for creating games. These tools can be used to create more complex games, but they offer only limited programming possibilities.

Many similar packages exist, several of which can be found at www.ambrosine.com/resource.html. One such program, Game Maker (www.gamemaker.nl), is a rapid-application-development tool currently used worldwide by young people at home and in schools to create two-dimensional and isometric games.

Figure 1 shows the Game Maker interface, which uses an object-oriented, event-driven approach. With Game Maker's drag-and-drop techniques, users can create games without writing a single line of code—but it also includes an interpreted programming language. The program produces stand-alone games that can be distributed freely; a version of Game Maker itself is available for free as well.

Game Maker has become extremely popular during the past few years. In 2003, users downloaded 1.7 million

copies of the program. Further, an active user group has a forum on which users post more than 1,000 messages daily (<http://forums.gamemaker.nl>). The youngest users, 8-year-olds, receive their introduction to computer programming through Game Maker. The oldest users, 80-year-old senior citizens, find a new hobby in the creation of games—and sometimes a new profession.

OBJECT-ORIENTED PROGRAMMING

In a computer game, everything is an object: the monsters, wall segments, coins, bonuses, power-ups, and the guns and bullets. Thinking about creating games means thinking about objects and how they react to one another and to the player's input. So the game creator naturally thinks in an object-oriented way.

To create a game using Game Maker, the designer creates objects. Some objects have a visual representation, such as an animated sprite. Others, like those that control game flow or maintain the score, might lack this feature. Multiple instances of the

same object can appear in the game at the same moment.

Instances have properties. Some are built-in, like the speed with which the instance moves and the sprite used to represent it. Others can be defined, manipulated, and checked using actions or code. The user must define each object's behavior. While some objects, like wall segments, will have no behavior, others, like the avatar representing the player, will most likely have complicated behavior.

Game Maker defines behavior in event-driven terms. Events occur for objects, and the designer specifies actions that the game must execute when these events occur. Typical events include object creation or destruction, user input, collisions between instances, alarm clocks, and events that happen at each step. To achieve this, the game designer can simply drag and drop actions into events, as Figure 2 shows. Game Maker has more than 100 built-in actions, ranging from moving the object in a particular direction to playing a sound or displaying a high-score list.

For more advanced tasks, the designer uses a code action to type in pieces of code that are executed when the event occurs. Within this code are close to 1,000 possible functions that can control all aspects of the game, including a particle system and network play functionality.

Inheritance, a powerful object-oriented programming concept, can be difficult to grasp in traditional applications. In game creation, however, understanding how inheritance works comes naturally. Take, for example, the well-known class of games based on *Breakout*, in which the user must destroy stones by hitting them with a bouncing ball. All stones exhibit similar behavior but will appear in a variety of shapes and colors. These characteristics make it logical and efficient to create one stone object and specify its behavior, then create similar objects with different colors that inherit the original stone object's behavior.

With Game Maker, users achieve this by simply setting the object's parent field. Some stones might have slightly different behavior—players might, for example, need to hit them twice to destroy them. To achieve this, users can simply override certain events and specify different actions for them. Users find this to be an extremely powerful mechanism that leads them intuitively into object-oriented programming.

MORE ADVANCED CONCEPTS

Once users become familiar with basic game creation, their interest usually turns to new areas such as behavior, artificial intelligence, or the creation of multiplayer games. Some want to create games with more elaborate graphics, such as those viewed from an isometric perspective. Others might become interested in how to simulate real-world physics, such as the motion of a car in a racing game.

This desire pulls the novice user deeper into the world of computing. Take, for example, artificial intelligence. Objects in a game require behavior. Even the ghosts in *Pac-Man* have some reasonably complex behavior. These enemies try to catch the player's character. Likewise, when the character eats a power-up pill, the ghosts flee in simulated terror. Further, the ghosts have differing behaviors: Some run toward the player while others guard specific areas.

Adding such behavior to a game requires some intuitive knowledge of rule-based systems and finite state machines. The ghosts, for example, can be in different states. Their behavior depends on their state and on information about the position of walls and the player. When particular events occur, the ghosts change state—for example, from hunter to hunted or from hunted to being dead—which leads to different behavior. Game Maker can easily demonstrate such concepts.

TEACHING GAME DESIGN

Game Maker is being used for teaching at all levels. Several organi-

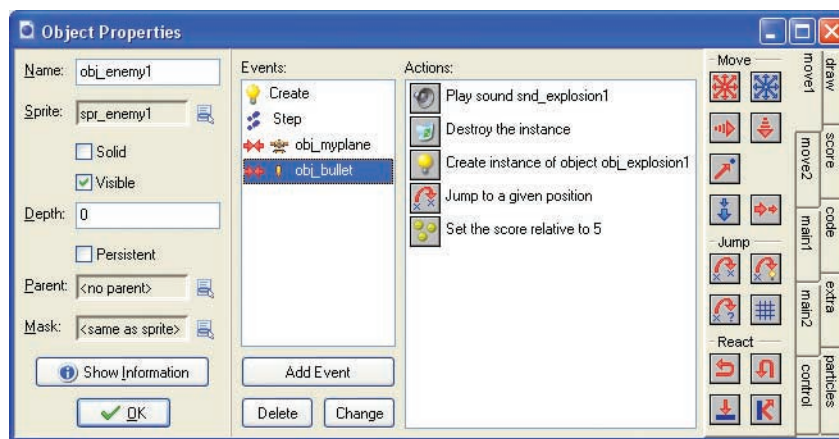


Figure 2. Object property form. The list of defined events for the enemy appears in the left center, while the actions that the game must perform when the enemy collides with a bullet appear to the right.

zations use the program in technology summer camps, most notably the Children's Technology Workshop (www.ctworkshop.com), which offers a complete camp curriculum based on the program and licenses its use to others. Several elementary schools have begun using Game Maker to stimulate student interest in computing.

Educators have increasingly incorporated game design into high school computer science curricula. Generally, students respond enthusiastically to these courses because they prefer programming in Game Maker to doing their usual assignments.

Game Maker also is used in the game design course taught at Utrecht University (www.cs.uu.nl). In this course, students first learn the basic concepts of designing games: how to set up the game's rules, describe the game in a design document, specify the game objects' behavior, and, most importantly, make a game fun to play. Later in the course, students explore three-dimensional game design. For more information about teaching game design with Game Maker, see www.gamemaker.nl/teachers.html.

People have made amazing games with Game Maker. To view a large repository of these creations,

see www.gamemaker.nl/games.html. More importantly, users enjoy creating the games and, while doing so, learn about programming and other, more advanced aspects of computer science.

Creating games appeals to all ages and to both males and females—as the many active female users of Game Maker demonstrate. Perhaps best of all, users of these tools are learning that creating games can be even more fun than playing them. ■

Mark Overmars is a full professor in the Department of Computer Science, Utrecht University and heads its Center for Geometry, Imaging, and Virtual Environments (www.give.nl). Contact him at markov@cs.uu.nl.

**Editor: Michael Macedonia,
Georgia Tech Research Institute,
Atlanta; macedonia@computer.org**