

As a youth taking my first computer steps, I remember how exciting it was to write a program that printed the first 100 prime numbers. However, today, computer programs have fancy interfaces that provide access to music, video and games; thus, those prime numbers' results no longer fascinate novice programmers. Instead, these initiatives want to create those 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, provides 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. But for today's users, spoiled by console and computer games, Logo is no longer flashy enough.

The "fun factor" of steering a virtual turtle can't possibly compare with the appeal of steering a real robot. This reality probably accounts for much of Lego MindStorms' success www.lego-mindstorms.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://bricx-cc.sourceforge.net/nqc/>.

The major 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 sensing, control loops and parallel tasks.

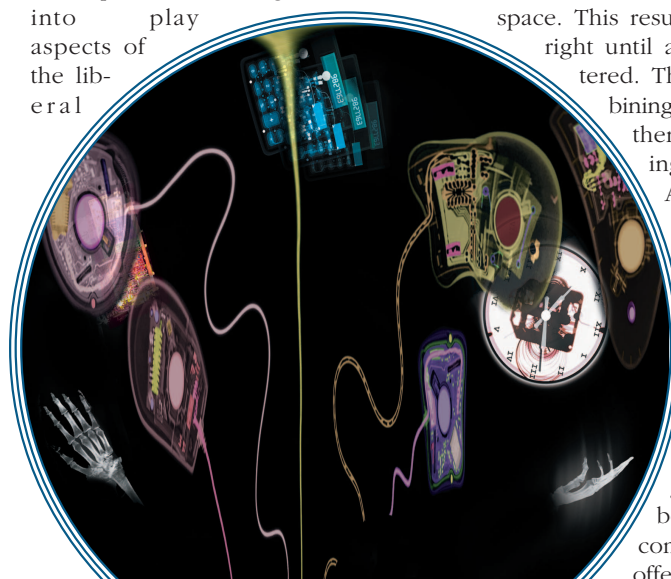
Game design

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



Learning object-oriented design by creating games

Mark Overmars

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 of 40 or more people. But, simpler alternatives—ones within the reach of students and hobbyists—do exist. Great games do not necessarily need fancy 3D graphics or huge game worlds. Budding game developers can have fun creating variations on *Pac-Man*, *Space Invaders* or simple platform games. And, with some creativity, they might even develop new game concepts.

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. Taking care of all the graphics processing, user interaction, collision detection, and timing is a huge task. Also, designing levels without an appropriate level editor is time-consuming. 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. This results in a motion to the

right until an obstacle is encountered. The user, through combining several of these rules,

then learns about designing rule-based behavior.

Although intuitive, this approach can create only rather simple games. Likewise, ClickTeam 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 written by the author, Game Maker www.gamemaker.nl, is a rapid-application development tool. Currently, young people around the world use it 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 the program 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.

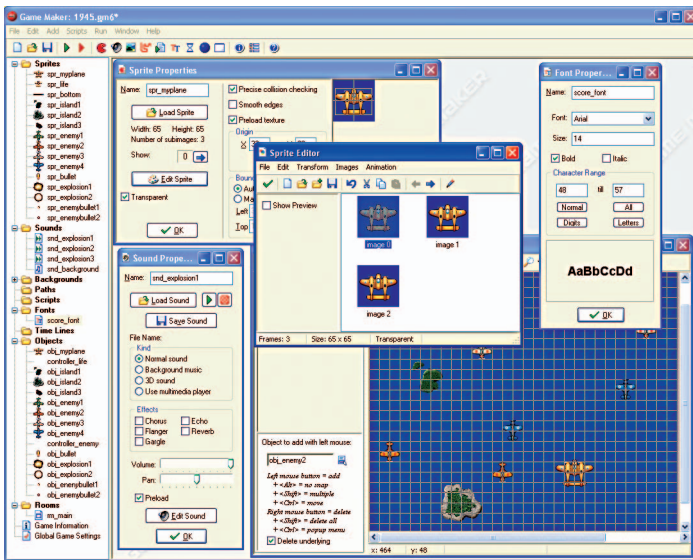


Fig. 1 The Game Maker interface. The left side displays resources, such as sprites and sounds, while the right side shows the sprite editor, room editor and other property forms.

Object-oriented programming

Many teachers indicate that it is difficult for students to understand object-oriented programming. This is somewhat surprising because object-oriented design is very natural. In real-life, we think in terms of objects with certain properties and behavior. Still, once people write a program they tend to adopt the traditional view of instructions being executed and control structures.

But when you are creating computer games, this mindset changes. In a computer game, everything is an object: the monsters, wall segments, coins, bonuses, power-ups, and the guns and the bullets. Thinking about creating a game 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 Fig. 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, there are close to 1,000 possible functions that can control all aspects of the game, including a particle system, network play functionality, and routines for three-dimensional graphics.

Inheritance

Inheritance, a powerful object-oriented

programming concept, can be difficult to grasp in traditional applications. The idea is that objects inherit certain properties and methods from other objects. They can partially overwrite this behavior. Inheritance is a powerful mechanism and experienced programmers often build large hierarchies of objects. For novice programmers this is somewhat overwhelming and it requires careful design as the behavior of an object is defined at multiple places.

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 a one stone object and specify its behavior. The person then creates similar objects with different colors that inherit the original stone object's behavior.

With Game Maker, users achieve this inheritance by simply setting the object's parent field. A child object inherits all behavior from its parent. Some stones might exhibit a 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 ability to be an extremely powerful mechanism that leads them intuitively into object-oriented design.

Artificial intelligence

Once users become familiar with basic game creation, their interest usually turns to new areas such as behavior and artificial intelligence. This desire pulls the novice user deeper into the world of computing. 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 of the playfield.

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 the walls and

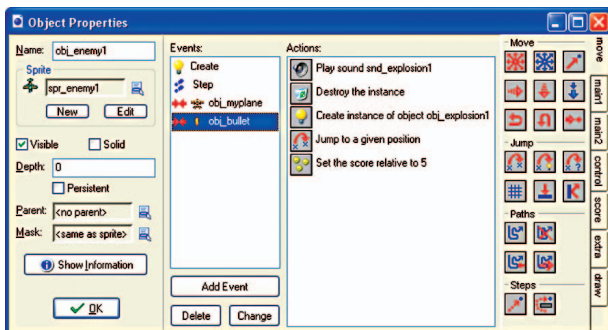


Fig. 2 Object property form. The list of defined events for the enemy appears in the left center, while the actions that the game must perform when the enemy collides with a bullet appear to the right.

the player. When particular events occur, the ghosts change. When the character eats a pill, the ghosts change from hunters to the hunted. When a timer runs out, they change back to hunters. And, finally when caught by the player, they turn from hunted to being dead. Such state-changes are easily shown by a finite-state diagram, in which the different states and state-

chronization. Again others might become interested in how to simulate real-world physics, such as the motion of a car in a racing game. The possibilities are endless.

Teaching game design

Game Maker is being used for teaching at all levels. Several organizations use the program in technology summer

or sought. And, after the programming, the game must be tested and tuned for best performance. Finally, documentation must be written. Creating a game makes an excellent group project in which students can combine their creativity and interests.

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 the game fun to play. Later in the course, students also explore three-dimensional game design but often they are more satisfied with their two-dimensional creations. For more information about teaching game design with Game Maker, see www.gamemaker.nl/teachers.html.

Finally

People have made amazing games with Game Maker. See Fig. 3 for some examples. Many more can be found on the Game Maker website. More importantly, users enjoy creating the games and, while doing so, learn about programming and other, more advanced, aspects of computer science.

Game creation appeals to all ages and to both sexes. Perhaps best of all, users of these tools are learning that creating games can be even more fun than playing them.

About the author

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

A shorter version of this paper appeared under the title *Teaching Computer Science through Game Design* in the April 2004 issue of IEEE Computer.

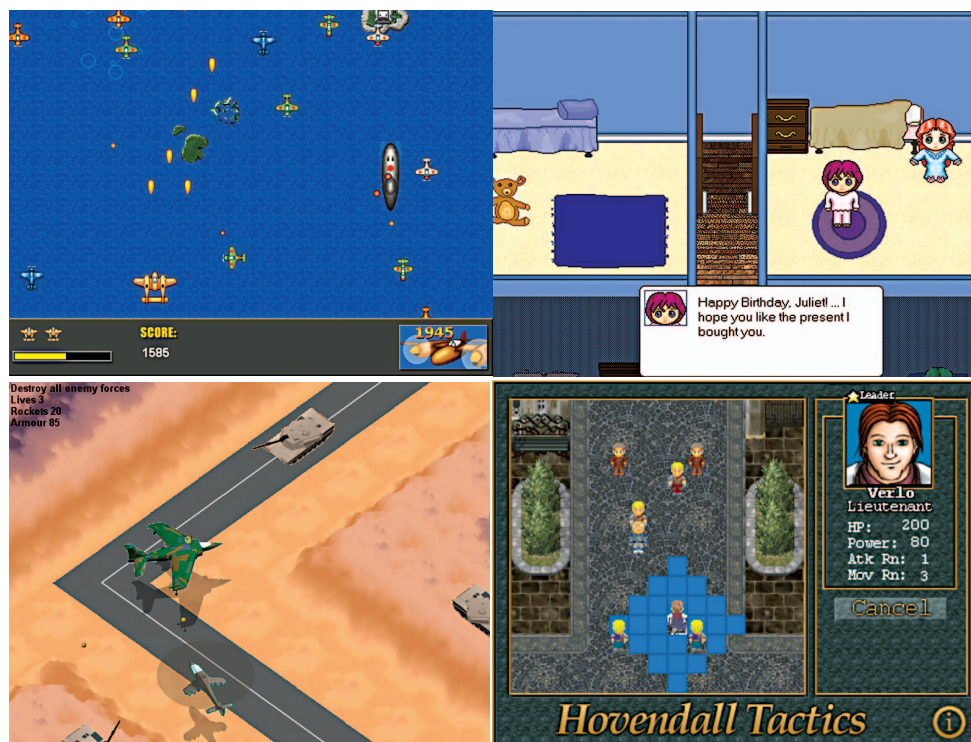


Fig. 3 A few examples of games created with Game Maker: a scrolling shooter, adventure game, isometric game and a turn-based strategy game.

changes are described.

Game Maker can easily demonstrate such concepts. For each state, a separate object can be defined with the corresponding behavior. When a state-changing event occurs, the ghost instance can change into a different object using a single action.

More advanced concepts

Step by step, the users will explore more advanced concepts of computer science. Some want to create games with more elaborate graphics, such as those viewed from an isometric perspective or full three-dimensional worlds. Or they start using particles systems to create fancy effects such as star fields, fireworks, flames and flying debris.

Others might become interested in creating networked games in which multiple players on different machines can play together. This will lead them into issues of message passing and syn-

chronization. Most notably, the Children's Technology Workshop www.ctworkshop.com, 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. But game design offers more than mere programming.

To design an effective game, students must think about the rules that define the game play. Such rules must be consistent and fair, requiring logical thinking and some probability calculus. A gripping storyline is often important for a game. The user-interface must be designed, and effective artwork and sounds must be created

Renew today
if you have not
already done so.