CHAPTER 11

Game Design

Chapter Preview

Section 11.1 **Introduction to Game Design**

Section 11.2 **Finalizing the Game Build**

Learning Outcomes

- ☐ LO 11.1 Describe the elements of a successful game.
- ☐ LO 11.2 Explain ways in which to provide game balancing.

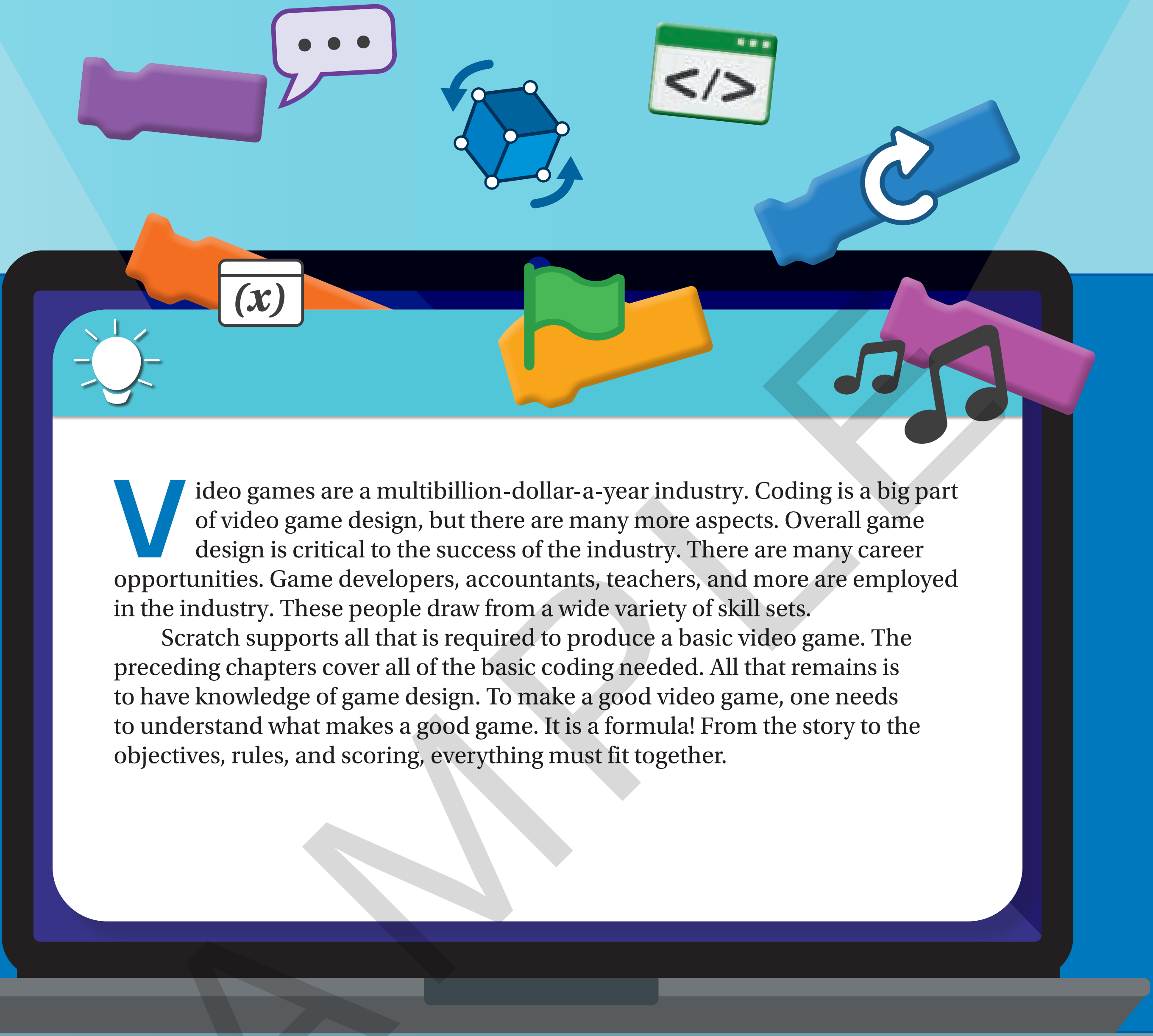
Essential Question

Why should video games be considered valid educational tools?

Terms

asset	game-design document	non-player character (NPC)
avatar	game loop	objective
boss character	game mechanics	player character (PC)
game balancing	genre	replayability

The CSTA icon notes correlations to the CSTA K–12 Computer Science Standards throughout this chapter. Please see Appendix C for a full listing.



Video games are a multibillion-dollar-a-year industry. Coding is a big part of video game design, but there are many more aspects. Overall game design is critical to the success of the industry. There are many career opportunities. Game developers, accountants, teachers, and more are employed in the industry. These people draw from a wide variety of skill sets.

Scratch supports all that is required to produce a basic video game. The preceding chapters cover all of the basic coding needed. All that remains is to have knowledge of game design. To make a good video game, one needs to understand what makes a good game. It is a formula! From the story to the objectives, rules, and scoring, everything must fit together.

College and Career Readiness

Reading Prep. Take the time to reread sections of this chapter that raise questions or cause confusion. Rereading can clarify the content and strengthen your understanding of the material.

LO 11.1

Describe the elements of a successful game.

Section 11.1 Introduction to Game Design

Recall from Chapter 8 that a video game has rules, a victory condition, an environment that enables play, and is software-based. A video game can be very involved, taking dozens of hours to complete. On the other hand, some video games are casual games. A casual game is one that can be completed in a short amount of time, generally less than eight hours.

The ***genre*** is the type or category of a game. There is not a set list of genres. The genre may be puzzle, action, side-scroller, cards, role-playing (RPG), or one of many more. A very famous game franchise that started out as a side-scroller game is the *Super Mario* series of games from Nintendo, as shown in **Figure 11-1**. In side-scroller games, the player character is centered on the screen, and the game world moves across the screen. The player character is seen from the side.

A basic rule of video game design is that a game should be easy to learn but difficult to master. Most games build on basic skills players have learned in other games while offering the player something new. In all cases, replayability is a key. ***Replayability*** means the game can be played over and over while maintaining the interest of the player. The player must want to play again and again.

Great video games are the result of thorough planning and careful execution. Elements of a video game that invite the player in are the characters and story. Game mechanics are the things the player can do and how the game reacts to the player. Ideally, an entire game is planned before any coding is done. Creating the game-design document is the first step in creating a video game.

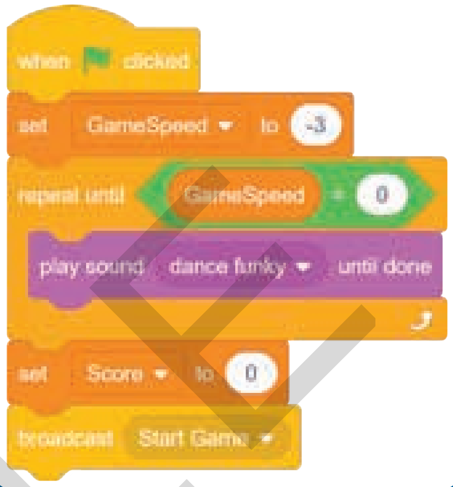
enchanted_fairy/Shutterstock.com

Figure 11-1.
The *Super Mario* game franchise is a famous example of side-scroller games.

Coding Conundrum

Sound Timing

A game-design document calls for music to start playing as soon as the game begins. The coder decided to use a when **Green Flag** clicked event to begin the music. Examine the code. What happens when this code is executed? How should it be corrected to start the music when the game begins?



```
when [flag] clicked
set GameSpeed ▾ to -3
repeat until GameSpeed = 0
    play sound dance funky ▾ until done
set Score ▾ to 0
broadcast Start Game ▾
```

Goodheart-Willcox Publisher

The music is not timed to the start of the game, which must be corrected.

Game-Design Document

A ***game-design document*** explains all details needed to create a game. This may also be called the *governing game-design document*. The genre, story, game mechanics, characters, audio, and algorithms are all described in detail in a game-design document. There is not a single format for the game-design document. The format depends on the game being developed. The document may be very basic for a simple game. For a large-scale production, the document may be hundreds of pages.

Typically, the game designer writes the document. Then, other team members use the details in the document to create the game. Coders, artists, musicians, voice actors, and others work as a team to create the game. Often a team of a hundred people develops a video game. Their stories are rich and powerful.

A completed game-design document is available on the student companion website at www.g-wlearning.com/informationtechnology/2260. This document details the side-scroller video game that will be created throughout this chapter named *Sidewalk Scramble*. Examine this document. Then, refer to it as you work through the chapter. This will show you how the document is constructed and how it applies to game development.

2-AP-18

FYI

A graphic organizer can be used to help develop the game-design document.

Story

Every great game has a story behind it. The backstory is everything that happened in the story before the start of the game. It is the history of the characters and the places in the game. Backstory is a key element in getting a new player to try the game. In addition, great games keep developing the story as the game goes along. A good developing story keeps the player coming back to the game.

Figure 11-2.
In this racing game, the player's avatar is a car.

Great game stories have the same qualities as the ones you write in English class. There is a hero. As you learned in English, this is the protagonist. There is also an antihero or villain, who is the antagonist. They each have flaws that make them interesting. One of the keys to a good story is conflict between the protagonist and antagonist. A good story has a climax, which is the crisis point when the conflict reaches its highest. Climax is followed by denouement, which is when the story concludes and all loose ends are tied up. These features come together to make a great story for a video game.

The ***avatar*** is the visual representation of the player character, as shown in **Figure 11-2**. Many games allow the player to choose between several avatars. This is common for online games since the player avatar is what other players see as the person who is playing the game. Allowing the player to choose an avatar helps the person get into the game. The player can choose an avatar that expresses their personality. The choice of avatar can also help connect the player to the story. For example, in a role-playing game (RPG) set in Medieval Europe, the player may have choices for avatars that include a knight in shiny or dark armor, a mystic, or a commoner.

In most games, there are other characters in addition to the hero and villain. There are enemies and friendlies. Enemies work with the villain to oppose the player. Friendlies work with the player to help achieve the game's goals and complete the story. Enemies and friendlies are computer-controlled characters.

The side-scroller game you will be creating in this chapter has a very basic story. The protagonist, Avery, is walking to the store. The antagonist is the boss. In video games, the ***boss character*** is the main enemy. When the boss is encountered, it must be defeated or the game is over. In this game, the boss character is a bat. Avery will encounter other game objects along the way that enhance the story.

FYI

When coding a game, apply what you have learned about clones and collision techniques to create the game mechanics.

Game Mechanics

Game mechanics describe how the game works. Think of them as the code segments that react to the player's input and keep the game running. For example, how a character reacts to key presses or cursor moves is a game mechanic. The game designers decide what the mechanic will specify. The scoring mechanism is another game mechanic. The control of all objects is dictated by the game mechanics. In Scratch, the stage is where the code for the main game mechanics is placed. Individual sprites also contain code for part of the game mechanics.

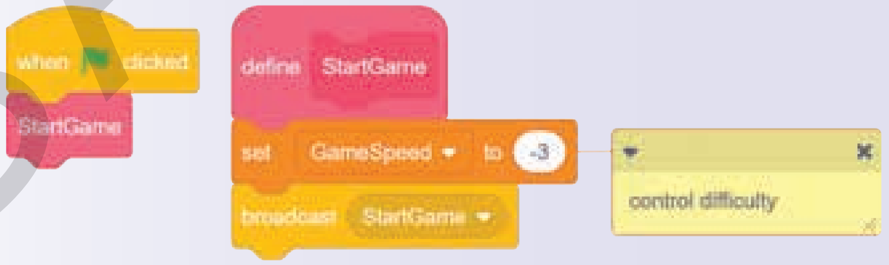
On occasion, a condition must be noticed and handled. This is part of the game mechanics. Game developers use Boolean expressions for this task. A variable can be made to store the value true when a condition is in effect. When the condition is no longer in effect, the variable is set to false. This variable can then be used to test a condition. For example, when a walking animation should play, a variable named walking can be set to true. When the walking should stop, the variable can be set to false. The conditional test is: **IF** walking equals true.

Hands-On Example 11.1A

Coding Game Mechanics

The basic story for the *Sidewalk Scramble* game is Avery must avoid obstacles as she walks to the store while collecting prizes. She must be able to jump over objects to avoid them. In this activity, the game mechanics begin to be set up.

1. Launch Scratch and delete the default sprite.
2. Choose the Wall1 backdrop from the library.
3. Applying what you have learned, create two new sprites from the Wall1 backdrop. Refer to Hands-On Example 10.4 if needed to review this process. Name the new sprites Wall01 and Wall02.
4. Delete the Wall1 backdrop. It is no longer needed.
5. Add code to each wall sprite that moves it to a starting location when the **Green Flag** button is clicked. One should move to (465, 0) and the other to (–15, 0). Other X coordinates can be used, but the sprites should fit exactly side-by-side. This ensures the sprites will be in the correct position to create the scrolling background.
6. Select the stage, and create a global variable named GameSpeed. This variable will be used to control the speed of all objects in the game. The speed of the objects determines the difficulty of the level.
7. Applying what you have learned, make a block for the stage named Start Game, and define it to set the GameSpeed variable to –3 and to send a broadcast named StartGame.
8. Add the Start Game custom block to a when **Green Flag** clicked event on the stage, as shown.



```
when [flag] clicked
StartGame
```

```
define StartGame
set GameSpeed ▾ to -3
broadcast StartGame ▾
```

```
control difficulty
```

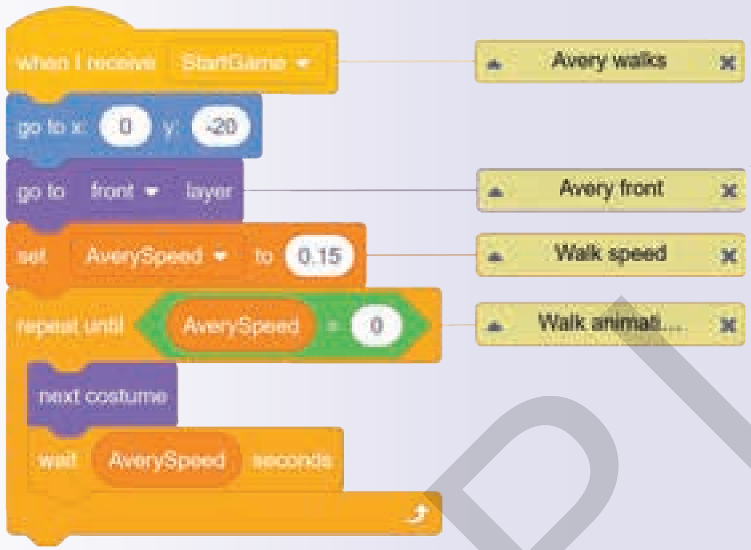
Goodheart-Willcox Publisher

9. Choose the Avery Walking sprite from the library.
10. Applying what you have learned, add a when I receive StartGame event block to the Avery Walking sprite.
11. Create a local variable named AverySpeed.

Continued

Continued

12 Applying what you have learned, add code to make Avery appear on the sidewalk in the left-to-right center of the stage and start walking, as shown. Set the AverySpeed variable to 0.15, and repeat costume changes until AverySpeed = 0. This allows for the walking to stop at the end of the game.

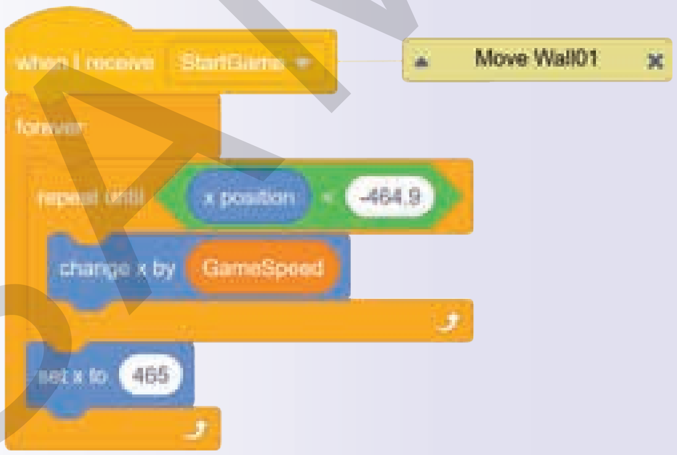


```
when I receive StartGame
go to x: 0 y: -20
go to front layer
set AverySpeed to 0.15
repeat until AverySpeed = 0
    next costume
    wait AverySpeed seconds
```

Goodheart-Willcox Publisher

13 Select one of the wall sprites, and add a when I receive StartGame event block.

14 Applying what you have learned, add code to scroll the wall across the stage from right to left, as shown. Refer to Hands-On Example 10.4 if needed to review this process. Avery will appear to be moving to the right, so the wall should move to the left.



```
when I receive StartGame
forever
    repeat until x position < -464.9
        change x by GameSpeed
    set x to 465
```

Goodheart-Willcox Publisher

15 Drag the code stack for moving the wall, and drop it onto the sprite thumbnail for the other wall. This copies the code to the other sprite.

16 Test the project. Correct any errors. If the wall does not wrap, try adjusting the value for the X position test.

17 Save the project as HOE11-01A_SidewalkScramble in your working folder.

Continued

Continued

Try It!

Write a simple story for a side-scroller game. Start a Scratch project, and set up the mechanics for the player character walking and the background moving. Choose a sprite from the library that has a walking animation or create your own sprite and costumes. Save the project as TI11-01A in your working folder. Create a game-design document to explain the details of your game, and save it as *LastName*_Side-Scroller in your working folder.

Objective

There needs to be a clear objective to the game. The ***objective*** of the game is the goal set for the player. Many games have several minor objectives and one main or overall objective. If the main objective is met, the player wins the game. Depending on the game, minor objectives may need to be completed to move forward. In some games, minor objectives provide a bonus or award, but do not need to be completed to finish the game.

Often, a side-scroller game has a set path for the player to follow, as shown in **Figure 11-3**. At the end of the path, there is a level-up. This is a minor objective. When the player achieves a level-up, the game becomes more difficult. Enemies may move more quickly or there may be more to avoid. The path may be more difficult to navigate. In many games, at the end of the final level, there is a "big boss" that must be defeated. Defeating the big boss is the main objective of the game.

In many games, the main objective is to achieve a high score. For example, scoring may be done by collecting certain objects while avoiding other objects. The rules for scoring must be clear for the user. If players cannot understand how to score points, they will quickly lose interest in the game.

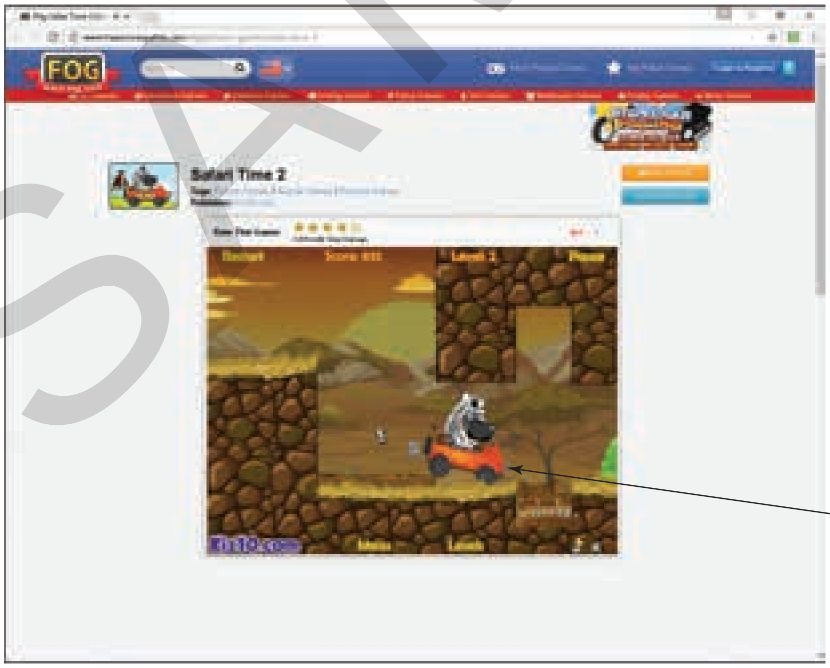


Figure 11-3.
Many side-scroller games have a path the player must follow to complete the game objective.

Math and Coding

Tracking Cards in a Game

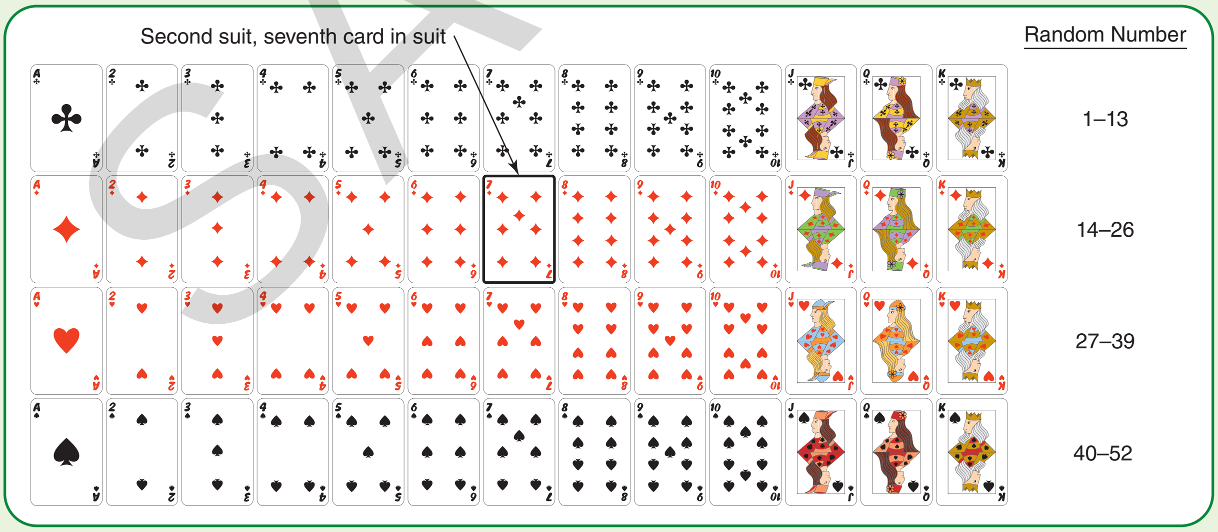
Thousands of video games fall into the genre of card games. There are many versions of the same games and a wide variety of different games, ranging in difficulty from Go Fish to Bridge. Every video game in the card games genre has at least one thing in common: it must keep track of the cards.

In a deck of cards, there are four suits and 13 cards in each suit for a total of 52 cards. The four suits are, in order, clubs, diamonds, hearts, and spades. Each card has one of the suits and one of the numbers on it. In order, the cards in a suit are ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, king. Depending on the game, the ace is either the lowest card in a suit or the highest.

To randomly select a card in a deck, generate a random number between 1 and 52. The algorithm for identifying which card that number represents requires a mathematical formula. Consider the 7 of diamonds. It is the twentieth card in the deck. Count the cards in the figure going from left to right, then down to the next row until you reach 20. You land on the 7 of diamonds. It is in the seventh position of the second suit. Applying this pattern, any card can be identified from the generated random number.

If the random number is 20, divide 20 by 13 (the number of cards in a suit). The result is a quotient of 1 with a remainder of 7. The remainder is the card's value number. Recall, the Scratch operator that finds the remainder is mod. For the ace, jack, queen, and king, the value number must be assigned a string. To compare the cards, use the value, which is a number. To identify the card, use the face value, which is a string. The integer quotient represents the card's suit. Number the suits from 0 to 3: clubs = 0, diamonds = 1, hearts = 2, and spades = 3. In Scratch, the integer quotient is found by applying the floor function after the division.

Create a Scratch project that generates a random number and tells the user what card has been drawn. Include the suit name and the face value. How could this function be used in a video game to show the user the card instead of telling the user which card was drawn? Save the project as MathCode11 in your working folder.



Goodheart-Willcox Publisher; silanti/Shutterstock.com

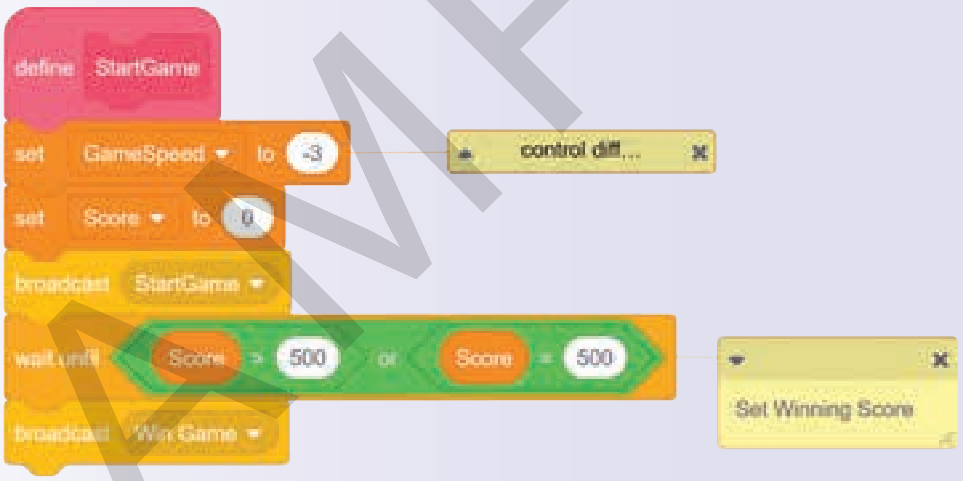
A complete set of playing cards consists of 52 cards. A video game must be able to keep track of the cards.

Hands-On Example 11.1B

Adding the Objective Mechanic

The main objective of the *Sidewalk Scramble* game is to reach the store before being caught by the boss. During gameplay, you will encounter a variety of enemies and friendlies. You must collect friendlies to score points and avoid enemies that will end the game. The store appears when Avery has scored at least 500 points. In this activity, you will create the mechanic for winning the game.

1. Launch Scratch, open HOE11-01A_SidewalkScramble, and save it as HOE11-01B_SidewalkScramble in your working folder. By creating stages or versions of the game, you can go back to an earlier state if something gets really messed up down the road.
2. Load the Urban backdrop from the library. This will represent the store when the game is won. During gameplay, it will be hidden behind the scrolling background walls.
3. Select the stage, and create a global variable named Score. Set this to be visible on the stage, and move it to the upper-left corner of the stage.
4. Initialize the Score variable to 0 in the Start Game block. This should occur before the broadcast is sent.
5. Add code to the StartGame block that will wait until the score is greater than or equal to 500 and then broadcast Win Game, as shown.



```
define StartGame
set GameSpeed to -3                          control diff...
set Score to 0
broadcast StartGame
wait until Score > 500 or Score = 500        Set Winning Score
broadcast Win Game
```

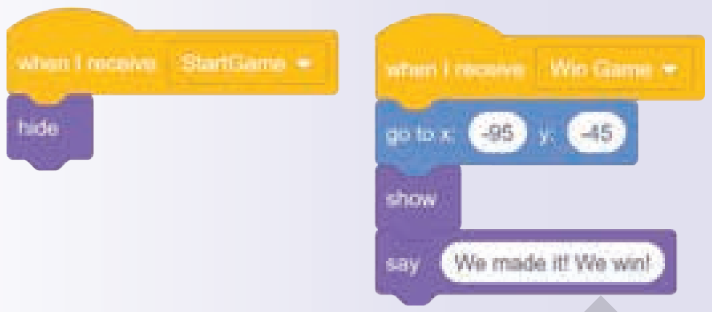
Goodheart-Willcox Publisher

6. Select the Avery Walking sprite. Add a when I receive Win Game event hat block. Add code to set the AverySpeed variable to 0. Also add code to hide the Avery Walking sprite.
7. Applying what you have learned, add code to both wall sprites to hide them when the Win Game broadcast is received.
8. Add the Avery sprite from the library. This will be displayed at the end of the game when the Avery Walking sprite is hidden.
9. Add code to the Avery sprite to hide the sprite when the Start Game broadcast is received.

Continued

Continued

10 Add code to the Avery sprite so that when the Win Game broadcast is received, the sprite is shown and says, “We made it! We win!” It is a good idea to also include code to position the sprite at an appropriate location, such as (–95, –45), as shown. Remember, this sprite will appear on the Urban backdrop, so it should be positioned where it will look good against this backdrop.



```
when I receive StartGame
hide

when I receive Win Game
go to x: -95 y: -45
show
say We made it! We win!
```

Goodheart-Willcox Publisher

11 Add an event to the stage that sets the Score variable to 500 when the space bar is pressed. This will be used to test the end of the game code. Shortcuts such as this are called *cheat codes* and are used by game studios to test small sections of the game.

Green Flag

12 Click the **Green Flag** button to test the game. When the space bar is pressed, the score should jump to 500, which simulates winning the game. The Avery Walking and the two wall sprites should be hidden, exposing the backdrop. The Avery sprite should be displayed and say, “We made it! We win!”

Stop Sign

13 Click the **Stop** button, and then click the **Green Flag** button. Notice the walls and Avery Walking sprites are still hidden. These sprites need to be initialized with a show block in the when I receive StartGame code stack. Make this change, and test the game again.

14 Save the project.

Try It!

Most side-scroller games have the objective of navigating a path to reach an end point. Scoring is usually part of the objective. Certain objects may increase the player’s score, while other objects may decrease the score or even end the game. Scoring allows for replayability where a player can try to achieve a higher score. Design an objective for your side-scroller, and update your game-design document. Open TI11-01A, and save it as TI11-01B in your working folder. Code the scoring and win mechanics for your game.

FYI

When designing characters, think about video games you have played. What did you like about the characters in those games? What did you dislike? Use this information to guide your own characters.

Characters

The ***player character (PC)*** is the character controlled by the human player. In multiplayer games, human players other than yourself are called *other player characters (OPCs)*. A ***non-player character (NPC)*** is any character in the game that is not controlled by a human. NPCs could be people, animals, or even objects like cars. For example, in a racing game, NPCs are other cars on the track. The characters are a key to the enjoyment of a video game. They add interest to the game. They can also help the player connect with the game.

Graphics for the sprite can make the character endearing, scary, mean, or display other qualities. The colors used in the graphic can tell the player the character is an enemy or a friendly, as shown in **Figure 11-4**. Determine the characteristics for each character and how it will behave in the game. Then, create graphics to match that behavior.

Many times the characters in a video game have special powers. For example, the player character may have the ability to become invisible. This allows the player to hide from enemies. Or, perhaps a character has ice-vision that allows it to freeze other characters in place. A character that is human may even be able to fly like a superhero.

The characters for the *Sidewalk Scramble* game are listed in **Figure 11-5**. Friendly game objects add points if they touch the avatar. Notice there is one friendly game object that must be avoided or points will be deducted. Unfriendly game objects, or enemies, deduct points from the score when touched by the avatar.

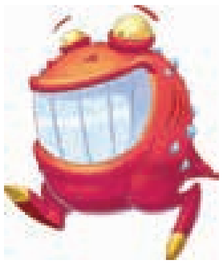

NextMars/Shutterstock.com

Figure 11-4.
The red color palette used on this character would tend to indicate it is an enemy. However, the style is not too dark or scary, which means the character is appropriate for a game aimed at younger players.

Name of Game Object	Type of Character	Role	Game Action
Avery	Player character (PC)	Protagonist; good at sports, loves animals, has power of invisibility	Walking, jumping, going invisible
Bat	Non-player character (NPC)	Antagonist (the boss character); blind, bites humans	Enemy; collision causes game over
Soccer ball	Non-player character (NPC)	Enter stage-right, bounces across to stage-left	Friendly; collision sends ball back and adds points
Dog	Non-player character (NPC)	Sits on sidewalk	Friendly; avoid collision to add points, collision loses points
Beach ball	Non-player character (NPC)	Enters stage-right, flies in air to stage-left	Friendly; collision sends ball back and adds points

Goodheart-Willcox Publisher

Figure 11-5.
The characters for the *Sidewalk Scramble* game and their roles.

Hands-On Example 11.1C

Coding the Character-Abilities Mechanic

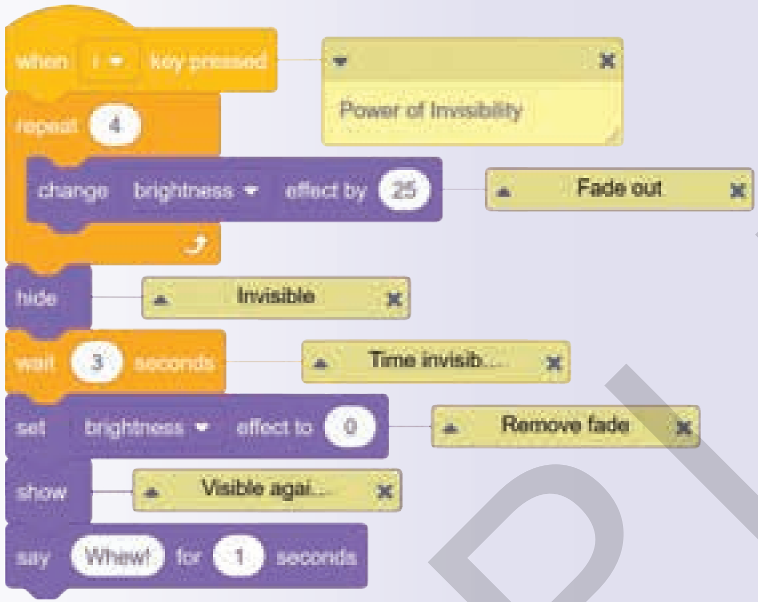
Avery must collect the two balls and avoid the dog to score points. Avoiding the enemy also scores points. If Avery touches the enemy, the game is over. To be able to do this, Avery must have the ability to jump. Also, she will be given the special power of invisibility to help her avoid obstacles.

1. Launch Scratch, open HOE11-01B_SidewalkScramble, and save it as HOE11-01C_SidewalkScramble in your working folder.
2. Applying what you have learned, add a when i key pressed event to the Avery Walking sprite. Pressing the [I] key will activate Avery's power of invisibility.
3. To add interest to the power of invisibility, use the brightness effect to lighten the costume before the sprite becomes invisible. Add a repeat 4 loop under the when i key pressed event, and add a change color effect by 25 block within the loop. Click the drop-down arrow in the block, and click **brightness** in the list. This loop will change the brightness of the costume from the default of 0 to a value of 100 in steps of 25 percent each.
4. After the repeat 4 loop, add code to hide the sprite and a delay of three seconds. The delay will give the enemy time to pass by Avery. During testing, you may find this value needs to be adjusted.

Continued

Continued

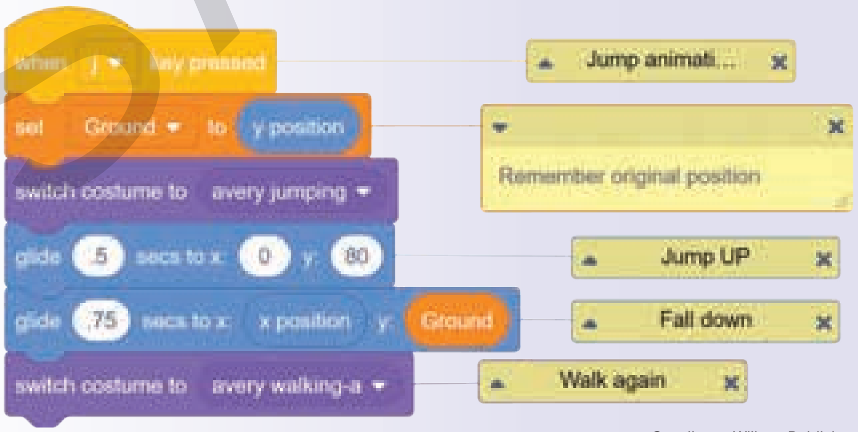
5. Add a set color effect to 0 block, and change the effect to **brightness**. Then, add code to show the sprite and code to have Avery say "Whew!" for 1 second, as shown.



```
when i key pressed                        Power of Invisibility
repeat 4
    change brightness effect by 25        Fade out
hide                                      Invisible
wait 3 seconds                            Time invisib...
set brightness effect to 0                Remove fade
show                                      Visible agai...
say Whew! for 1 seconds
```

Goodheart-Willcox Publisher

6. Applying what you have learned, add a new costume to the Avery Walking sprite for jumping. Alternately, you can save the costume created in the previous chapter and import it as a new costume, but depending on how you saved it, you may need to remove the white background.
7. Applying what you have learned, modify the code for the walking animation so the jumping costume is not included. There are several ways to do this, such as using the costume # variable in a test to see if the current costume is the jumping costume.
8. Applying what you have learned, add a when j key pressed event to the Avery Walking sprite. Pressing the [J] key will make Avery jump.
9. Applying what you have learned, add code for the jumping animation under the when j key pressed event. To ensure Avery returns to the same location on the sidewalk, store the Y coordinate in a variable before she jumps, as shown. When she finishes jumping, display the first costume in the walking animation.



```
when j key pressed                                    Jump animati...
set Ground to y position                              Remember original position
switch costume to avery jumping
glide .5 secs to x: 0 y: 80                           Jump UP
glide .75 secs to x: x position y: Ground             Fall down
switch costume to avery walking-a                     Walk again
```

Goodheart-Willcox Publisher

Continued

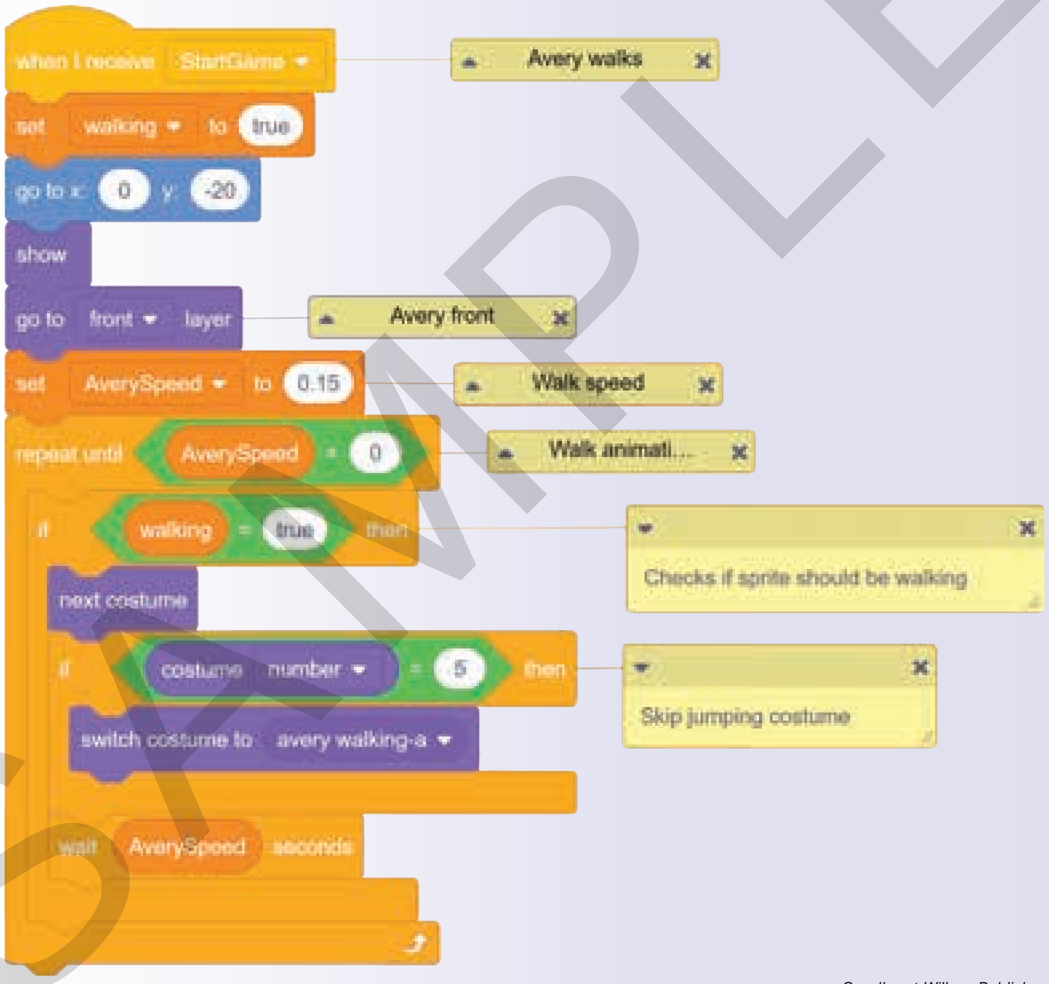
Continued

Green Flag

10 Test the project. Check that Avery uses her power of invisibility when the [I] key is pressed. Also, use the [J] key to make her jump. Notice that there is a problem with the jumping. The jumping costume is only displayed for a fraction of a second, and then she walks in the air for the rest of the jump. This needs to be fixed.

11 Make a local variable for the Avery Walking sprite named Walking. When Avery is jumping, set this variable to false. When the jump is complete, set the variable to true.

12 Modify the walking code to include a test for **IF** Walking = true, as shown. The walking animation should be displayed only when this test is true. Also, be sure to initialize the Walking variable to true at the beginning of the game.



```
when I receive StartGame            Avery walks
set walking to true
go to x: 0 y: -20
show
go to front layer                   Avery front
set AverySpeed to 0.15              Walk speed
repeat until AverySpeed = 0         Walk animati...
    if walking = true then          Checks if sprite should be walking
        next costume
        if costume number = 5 then  Skip jumping costume
            switch costume to avery walking-a
        wait AverySpeed seconds
```

Goodheart-Willcox Publisher

13 Test the project. The jumping costume, and only that costume, should be displayed throughout the entire jump. The jumping costume should not be displayed during the walking animation.

14 Save the project. The mechanics for the player character are complete. The scoring mechanics will be included in the obstacles.

Continued

Continued

Try It!

Consider the game objective you decided on earlier for your side-scroller game. What special abilities will be needed to complete the objective? Will the character fly, have a super-springy jump, or be able to put up a force field? Decide on the abilities for the player character, and update your game-design document. Open TI11-01B, and save it as TI11-01C in your working folder. Create and code the moves for the player character.

LO 11.2

Explain ways in which to provide game balancing.

Section 11.2 Finalizing the Game Build

Finalizing the game build brings everything together into a completed game. Assets are collected and added to the game. Game elements are programmed. The game is thoroughly tested, and the gameplay is balanced.

Collecting Assets

An ***asset*** is any audio, graphic, video, or text that is used in a video game. It is anything in the game that is not code. Game objects such as characters, tokens, obstacles, and on-screen instructions are assets. So, too, are the background music and the sound effects that play for game events.

In large development teams, a director uses the game-design document to guide the project. This includes identifying which assets are needed. A project manager works under the director. This person is responsible for guiding the team members through the completion of all tasks. Coders are not the only ones on the team. Artists, composers, and animators are also on the team. These team members create or collect the game assets.

There are several ways to collect assets in advance of using them in the game. Original assets can be created by the development team. Existing assets can be used from a previous project. Assets can be acquired from online sources. Remember to obtain the proper permissions to use these assets. In many cases, an asset is licensed from an asset developer and a fee paid for its use. For example, songs are licensed from recording artists, studios, and recording companies.

Libraries are usually set up to hold assets. A library is a collection of the asset files. As you have seen, Scratch has libraries for sprites, backdrops, and sounds, as shown in **Figure 11-6**. When the coder is ready to incorporate an asset, the library is the place to go to find it.

FYI

Think about how many songs games like *Rock Band* or *Guitar Hero* contain. Each song had to be licensed from the artist or record company that owns it.

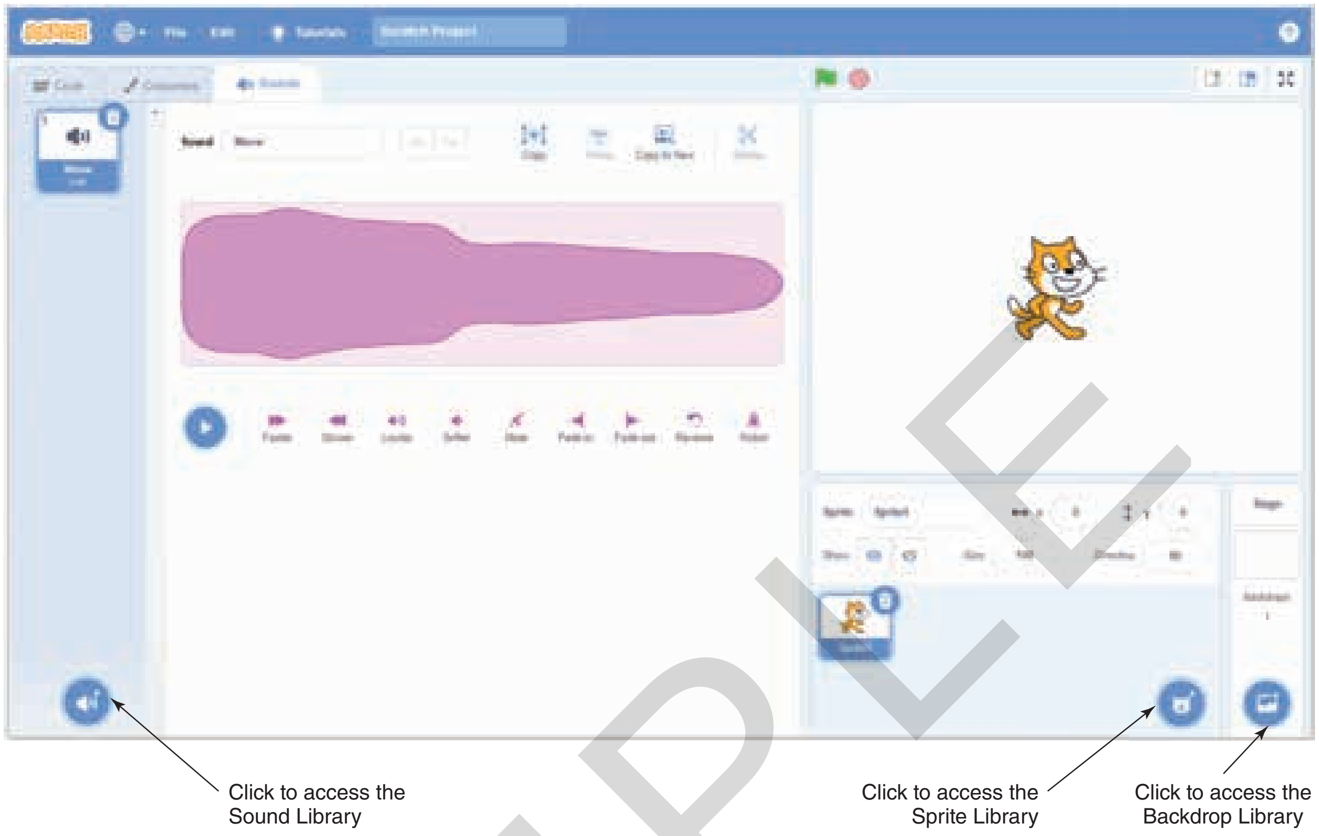

Click to access the Sound Library
Click to access the Sprite Library
Click to access the Backdrop Library
Goodheart-Willcox Publisher

Figure 11-6.
Assets are usually stored in libraries for easy access. Scratch contains libraries for sprites, backdrops, and sounds.

The game should look well thought-out and coordinated. The sprites and audio contribute to the overall look and feel of the game. They should be carefully selected to match the game's theme.

As mentioned earlier, the color of the sprites can provide visual cues. Bright colors tend to convey a happy theme. Dark colors generally create a more sinister theme. Choose colors for sprites that will help set the tone of the game. However, do not rely on color alone. More than 10 percent of the population is color-blind.

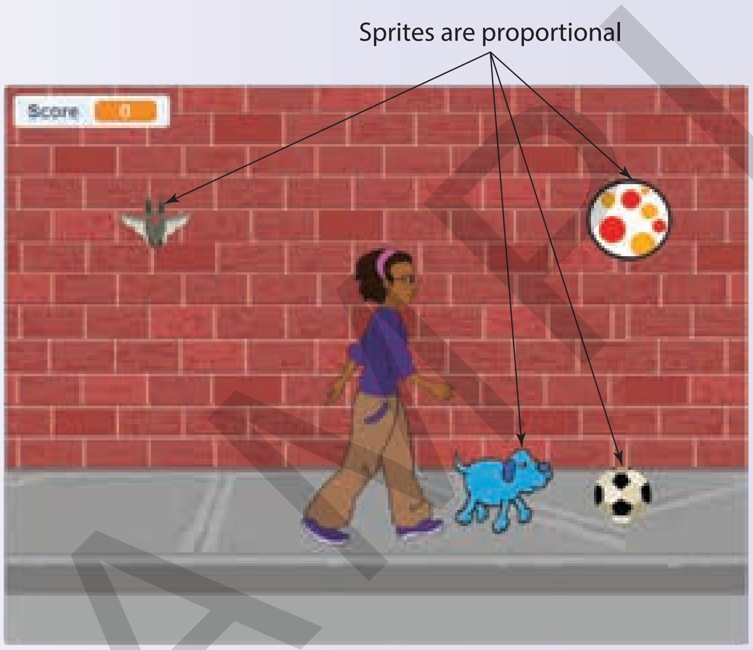
The background music plays a large part in setting the mood of the game. In many side-scroller games, the background music is happy and upbeat. However, if the game has a darker theme, the background music should not be happy. Instead, select music that is darker or slower. Similarly, sound effects for interactions should match the type of interaction. A good interaction should have a pleasant sound. A bad interaction should have a sound that seems negative.

Hands-On Example 11.2A

Adding Assets

The *Sidewalk Scramble* game has two assets for tokens, one asset for a friendly, and one asset for an enemy. In addition, sound effect assets for the collisions and background music are required.

1. Launch Scratch, open HOE11-01C_SidewalkScramble, and save it as HOE11-02A_SidewalkScramble in your working folder.
2. Applying what you have learned, load these sprites from the library: Soccer Ball, Beachball, Bat, and Dog2. The balls will be tokens the player collects. The bat (enemy) and dog (friendly) must be avoided by the player.
3. The sprites need to be resized to fit with the scale of Avery and the wall, as shown. Change the size percent to resize each sprite as needed. You may decide to resize Avery as well.



Goodheart-Willcox Publisher

4. Select the Soccer Ball sprite, and click the **Sounds** tab. Preview the basketball bounce sound already attached to the sprite. This will work for the collision between Avery and the ball.
5. Select the Beachball sprite, and delete the pop sound attached to it. Then, load the Kick Drum sound from the library. Having different sounds for the two balls will help the player know which one was contacted.
6. Select the Dog2 sprite, and preview the dog1 sound already attached to it. This will work for the collision if Avery cannot avoid the dog.
7. Applying what you have learned, assign the Screech sound from the library to the Bat sprite.

Continued

Continued

8. Select the stage and assign the Xylo1 sound or other upbeat, happy sound from the library. Alternatively, apply what you learned in Chapter 9 to compose a piece for background music.
9. Save the project.

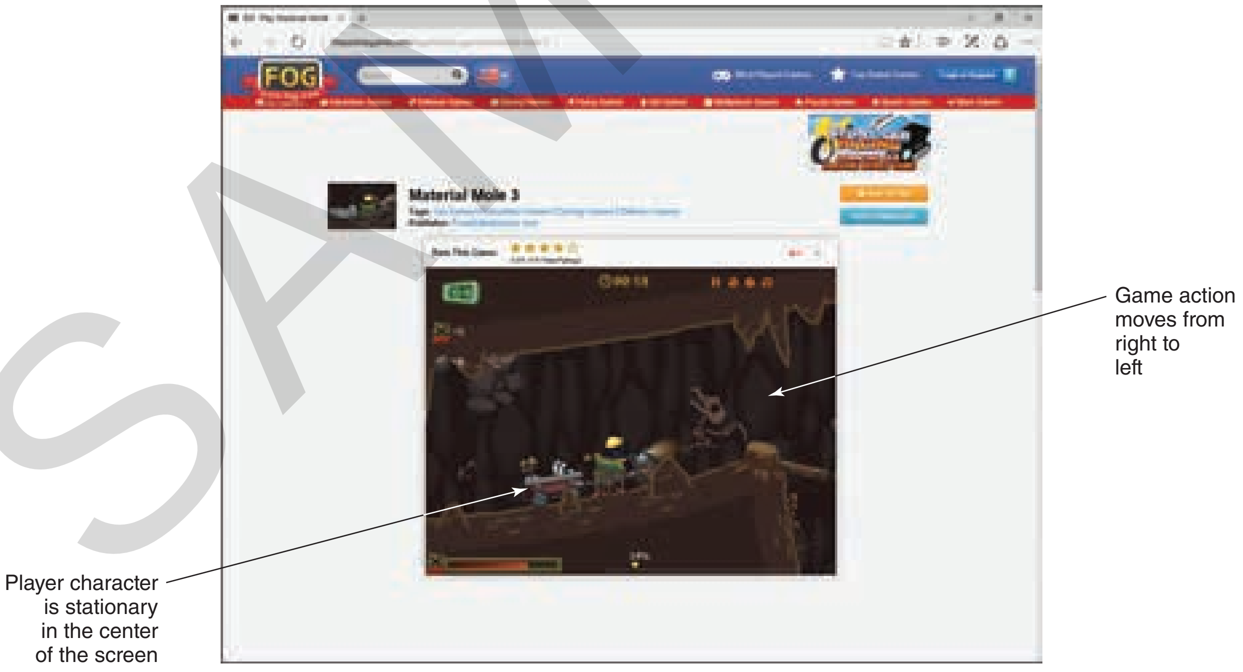
Try It!

Decide what game assets are needed for your side-scroller game. Will the player character collect tokens or other items? These need to be added as assets. Consider what sound effects are needed for interactions. Select background music for the game. Update your game-design document. Open TI11-01C, and save it as TI11-02A in your working folder. Add the game assets you have decided are needed for the game.

Game Elements

Some assets become game elements, such as tokens or enemies. The success of the game can depend on how game elements are included. If too many elements are onstage at once, the player can get confused or overwhelmed. If there are too few, the player can get bored.

In a side-scroller game, the player character usually appears to be moving from left to right, although the character stays in the middle of the screen, as shown in **Figure 11-7**. Most game elements travel from right to left.


FOG
Material Mole 3
Game action moves from right to left
Player character is stationary in the center of the screen

www.freeonlinegames.com

Figure 11-7.
In most side-scroller games, the player character remains in the middle of the screen but appears to be moving to the right.

On occasion, an enemy may sneak up from behind the player character (from the left). However, all elements must handle collision with the player character and have an exit strategy. What happens when the game element touches the player character? What happens to the game element once it passes stage left? An algorithm must be developed to explain what should happen.

There can be much repetition in a game. Launch a game element. Process the game element. When the game element is done, repeat. The process of repeatedly launching and processing game elements is known as the ***game loop***. Often, a process called *stub programming* is used to keep the game loop running while the code is being developed. Critical parts of a game element are added to keep the game loop running without fleshing it out completely. This allows testing on other parts of the game without stopping because the program cannot get past an element.

In Scratch, control of game elements should be through the stage. The stage is a central location, which makes it convenient to organize code for game elements. Broadcasts can be sent from the stage to the individual elements to start their unique instructions.

Hands-On Example 11.2B

Developing the Game Loop

In the *Sidewalk Scramble* game, the game elements will be generated as clones on a random basis. Only one game element will be onstage at a time.

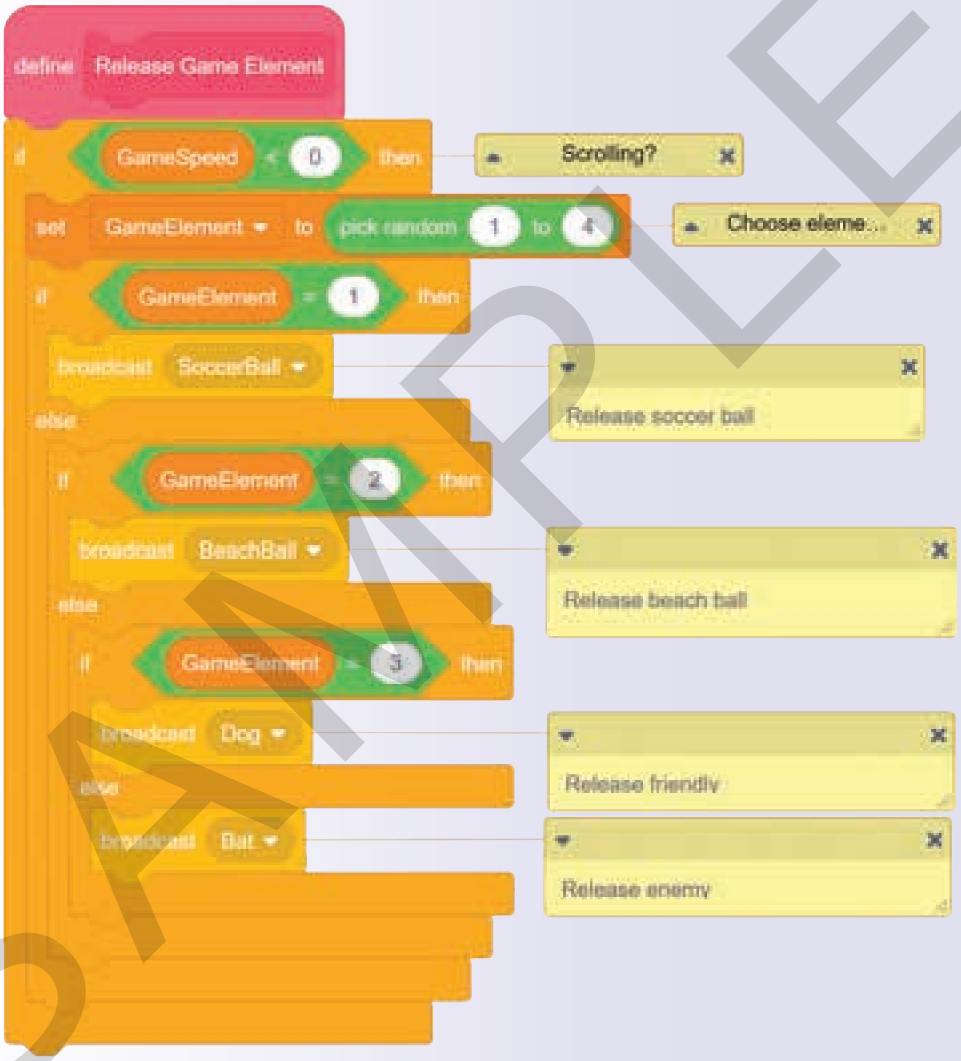
1. Launch Scratch, open HOE11-02A_SidewalkScramble, and save it as HOE11-02B_SidewalkScramble in your working folder.
2. Applying what you have learned, add code to the stage to play the background music. It should only play while there is scrolling. In the Start Game custom block, after the code to set the GameSpeed variable, send a broadcast named Start Background Music.
3. Add a when I receive Start Background Music event block to the stage. Add a repeat until GameSpeed = 0 loop to the event. Within, add a play sound xylo1 until done block to the loop. Since the GameSpeed variable controls scrolling, the music will only play when there is scrolling.
4. Modify the cheat code so that when the space bar is pressed, the GameSpeed variable is set to 0 as well as the Score variable set to 500.
5. Test the project. Check that the background music stops playing when the space bar is pressed. Note: since the sound is set to play until it is done, the sound may continue for a few seconds after the space bar is pressed.

Green Flag

Continued

Continued

6. Make a block named Release Game Element, and add it to the beginning of the define Start Game code stack for the stage. This block will control when a game element is added to the stage.
7. Define the Release Game Element block to randomly select a game element to release. There are four game elements, so generate a random number between one and four. Branch on the value of the random number to broadcast which game element is to be launched, as shown. Use broadcasts to signal the release of the elements. The actual release will be coded in each sprite.



```
define Release Game Element
if <(GameSpeed) < (0)> then                                  // Scrolling?
    set [GameElement] to (pick random (1) to (4))            // Choose eleme...
    if <(GameElement) = (1)> then
        broadcast [SoccerBall]                               // Release soccer ball
    else
        if <(GameElement) = (2)> then
            broadcast [BeachBall]                            // Release beach ball
        else
            if <(GameElement) = (3)> then
                broadcast [Dog]                              // Release friendly
            else
                broadcast [Bat]                              // Release enemy
```

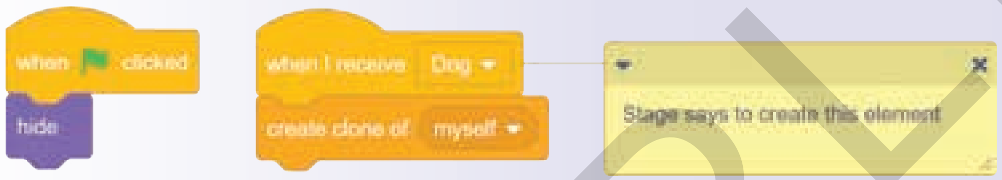
Goodheart-Willcox Publisher

8. Add an event to the stage for when the Game Element Done broadcast is received. Note: a new broadcast can be named directly in the "when received" event hat block. Under this block, add a Release Game Element block. When each element is done, it will send this broadcast, and the stage will signal to release a new element.

Continued

Continued

9. Select the Soccer Ball sprite. Add codes so that when it receives its message, the Game Element Done broadcast is sent. This stub programming completes the game loop, which will keep the game running even if a particular sprite has not been fully coded yet. Repeat this for the Beachball, Bat, and Dog2 sprites.
10. Applying what you have learned, add code to the Dog2 sprite to create a clone of itself when it receives its message from the stage. Also, add code to hide the sprite when the **Green Flag** button is clicked.
11. Applying what you have learned, add code to animate the clone, react to a collision with Avery, and change the score as appropriate, as shown. The score change should be either +25 for a good event or –25 for a bad event. In this case, Avery should avoid the dog, so a collision should deduct 25 from the score. Include a test to see if the score is 500 or more, and send the broadcast Win Game if so.



```
when [flag] clicked
hide

when I receive [Dog ▾]
create clone of [myself ▾]
```

Stage says to create this element

Goodheart-Willcox Publisher

12. Similarly, code the beachball and soccer ball. Start the beachball over Avery's head so that jumping to touch the ball adds points. Roll the soccer ball along the sidewalk at her feet so that a collision adds points.
13. Applying what you have learned, code the bat to appear on the left side of the stage and move to the right, flapping its wings as it goes. Multiply the GameSpeed variable to a negative value to make the sprite move to the right. Play the Screech sound when the clone is created to alert the player the enemy is arriving. If Avery misses the bat, add points to the score. If the bat touches Avery, keep the clone and send the broadcast Game Over.
14. When the stage receives the broadcast Game Over, set the Game Speed variable to 0. Also, use the stop all sounds block to silence the background music. When the Avery Walking sprite receives the Game Over broadcast, set the AverySpeed variable to 0. Also, display a message to the player indicating the game is over.
15. Test the game. Make sure all aspects are working correctly, from the animations to the scoring, winning scenario, and game over scenario. Save the project.

Try It!

Open TI11-02A, and save it as TI11-02B in your working folder. Code all of the interactions and scoring needed for your side-scroller game. Refer to your game-design document. This is your guide for creating the game. If you find something in the document does not reflect how it must happen in the game, be sure to update the document.

Game Balancing

2-CS-01, 2-DA-09, 2-AP-15, 2-IC-22

Many decisions are made when developing the algorithms and coding the project. Hopefully, these decisions will lead to a game with good gameplay. It is only through testing and game balancing that the best gameplay can be created, as shown in **Figure 11-8**. ***Game balancing*** is the process of evaluating the gameplay and making adjustments to ensure the game is playable and interesting. This is similar to writing a story or report where you edit, revise, and rewrite until it is finalized.

Developers play the game after each addition to see how fun the game is. Often, outside game testers are invited to play a game or parts of a game during development. They are asked to describe the gameplay experience and make suggestions for how the game could be more fun. Based on feedback from the testers, coders adjust the game. Then, the game undergoes more testing.

Recall, a game should be easy to learn, but hard to master. Allow for learning and skill-building on the part of the player. As the player moves to higher levels, the game should become harder. But, a player must have a reasonable expectation that the game is winnable. Fix anything in the game that continually produces more losses than wins.

ESB Professional

Figure 11-8.
Developers have testers play the game to find bugs and areas to improve.

Science and Coding

Gravity in Video Games

In many video games, objects fall from the top of the stage to the bottom. Using precise physics, the object should fall using Sir Isaac Newton's formula for gravity. The object should appear to accelerate as it approaches the bottom of the stage. The distance an object falls due to the force of gravity is determined by the formula:

$d = 1/2\, gt^2$

where *g* is the value for the force of gravity and *t* is the amount of time in seconds the object has been falling. Near the surface of Earth, the value of *g* is 32 feet per second squared. The formula is simplified to $d = 16t^2$. The table shows the distances for the first four seconds of a fall.

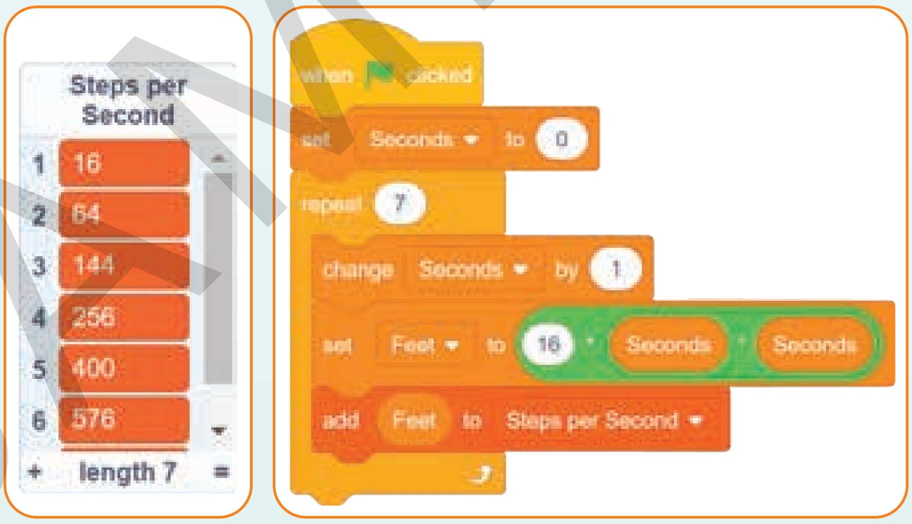
Seconds	Distance (feet)
1	16
2	64
3	144
4	256

Goodheart-Willcox Publisher

Suppose an object is dropped from the top of the stage. In terms of the action of a video game, one second is a very long time. Unless the scale of the stage is very big, the acceleration due to gravity will be less important. In a game that has a theme of Pennies from Heaven, the scale is large enough. The first verse of this song is:

Every time it rains, it rains pennies from heaven.
Don't you know each cloud contains pennies from heaven?
You'll find your fortune's fallin' all over the town.
Be sure that your umbrella is upside down.

The game mechanic drops pennies from clouds. As they fall to Earth, the player character must run back and forth trying to catch them in an umbrella. If the clouds are at 800 feet, the pennies take about seven seconds to fall. Calculate the distance each penny falls each second. Use Scratch to make the calculation, as shown.



	Steps per Second
1	16
2	64
3	144
4	256
5	400
6	576
+	length 7 =

```
when [flag] clicked
set Seconds to 0
repeat 7
    change Seconds by 1
    set Feet to 16 * Seconds * Seconds
    add Feet to Steps per Second
```

Goodheart-Willcox Publisher

This code calculates the distance traveled for the first seven seconds of fall from 800 feet.

In Scratch, use a "glide to" block to move each penny to the total distance each second. This does not exactly replicate real-world physics. In real-world physics, there is a continuous increase in speed during the seconds that pass. That requires many calculations. In a video game, "pretty good physics" can be used to save on calculations as long as the game provides the illusion of correct motion.

Create a Scratch project to simulate pennies falling from the clouds. Include a player character that can be moved back and forth to collect the pennies as they fall. When testing the game, notice the jumpiness in movement that occurs at the seconds boundaries. How could the code be improved to remove this jumpiness?

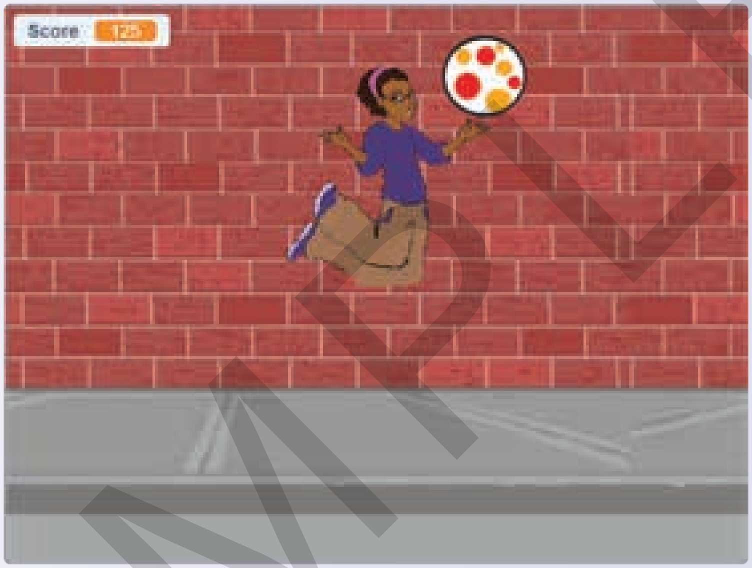
Hands-On Example 11.2C

2-CS-01, 2-IC-22

Balancing the Game

It is time to complete the *Sidewalk Scramble* game. All of the assets and coding are in place. Now, the game must be balanced. These steps are suggestions for improving the game. Determine if they are good improvements, and implement them if they are.

1. Launch Scratch, open HOE11-02B_SidewalkScramble, and save it as HOE11-02C_SidewalkScramble in your working folder.
2. Play the game from beginning to end, as shown. Take notes on anything that needs to be fixed or adjustments that could be made so the game is more enjoyable. Then, apply all of the skills you have learned to this point to balance the game.



Goodheart-Willcox Publisher

3. If winning is too hard, lower the winning score. Create a global variable named Win Score, and insert the variable wherever the value is currently hard-coded as 500. Then, add code to initialize the variable to a lower value when the game starts. Having the winning score set with a variable makes it easier to adjust the value to tune the game.
4. If the game goes too slowly, decrease the value for the GameSpeed variable, and increase the value for the AverySpeed variable.
5. If the same pace is boring, create a timer to fire every 10 seconds, and increase the value of the GameSpeed variable when the timer fires. This can be considered a form of leveling up, as the difficulty will increase as the player moves through the game.
6. If the bat flaps its wings too rapidly, add a wait block in the animation. Adjust the value to create an animation that appears to match the speed at which the bat is flying.
7. If the bat moves too quickly to the right, it may be too difficult for the player to avoid it. This can be adjusted by changing the speed of the bat or by moving the position of the player character farther to the right on the stage.
8. To add interest to the game elements, change their paths from straight lines to a sine wave.
9. Bounce the balls back off stage right after a collision using a glide block. This will require adjusting the test for when the game element is off the stage to include either the left or right side. The starting position of each element may also need to be adjusted.

Continued

Continued

10. If the sound is too soft or too loud, adjust the volume. This can be done through code, or the sound can be edited on the **Sounds** tab.
11. Evaluate the sound effects to see if they make sense with the game action. If not, change the sounds.
12. One way to add skill-building to a game is to include instructions for the player. For example, the first time the soccer ball appears, it can say "Kick me to get 25 points, but if you miss me you will lose 25 points." Consider instructions to add to the game, and then code these for the first instance of each game element.
13. If the jump is not fun, speed it up or slow it down.
14. If the game is over too soon, add a LevelUp custom block that is called at a certain score level or elapsed time. In the definition for this block, increase the game speed. Increase the value of the AverySpeed variable to keep pace with the background. Add a variable for the score changes and make it a larger or smaller number at different game levels. Adjust the jump speed at different levels.
15. Make any other adjustments you identified when testing the game.
16. Save the game. This is the final version of the game.

Try It!

Open TI11-02B, and save it as TI11-02C in your working folder. This will be your final game. You need to test your game and note anything that needs to be balanced. Ask a classmate to be a tester for the game. Consider the suggestions your classmate makes for improving the game. If any changes you will make differ from the specifications in the game-design document, update the document. Then, update the game.

Cooperative Coding

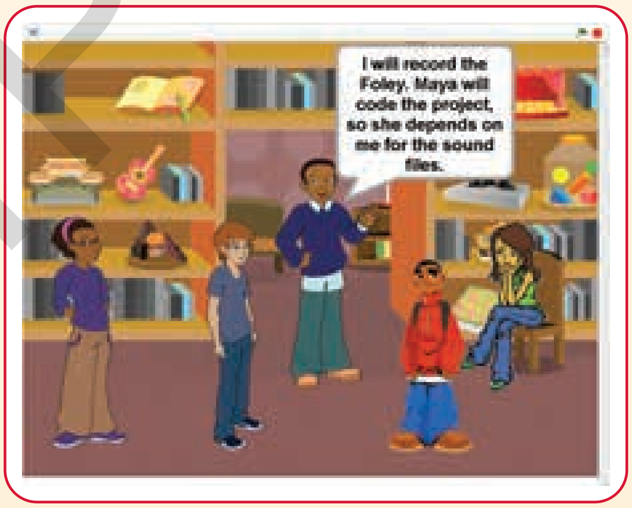
2-AP-18, 2-IC-22

Game Development

The development of a video game is a prime example of cooperative coding. Game-development teams are common in the industry. These teams are made up of artists, audio developers, video developers, writers, game designers, testers, coders, and others. The artists create graphics for game elements and characters. For some games, the graphics are 2D. Other games have 3D graphics. Animators use the art assets to create characters and elements with motion. Audio developers produce the Foley and the music. Recall, Foley is everyday sound effects. Video developers create video snippets that advance the story. Writers develop the characters and write the dialog as well as the backstory for games. Each of these people must have at least minimal coding skills to accomplish their jobs.

Jacob Lund/Shutterstock.com

Collect a team of students to serve as a game-development team. Brainstorm ideas for a video game with the entire team. Then, select one idea. Assign each team member a role in the design of the game. Each team member should think about the game in relation to their assigned role. Then, write one or two sentences about the following items.


I will record the Foley. Maya will code the project, so she depends on me for the sound files.
Goodheart-Willcox Publisher

- Describe what activities will be part of your role.
- Identify what other team members you depend on and for what.
- Identify the team members who depend on you to get their jobs done and how.

Create a Scratch project to show the various team members and their roles. Create a new sprite for each member. Use Scratch to record audio for each team member saying their responses to the above items. Be sure to create sound for the appropriate team member's sprite. Display all the team member sprites on stage. Create mouse click events for each sprite that plays the audio made by the team member. Include displayed text for accessibility for those with hearing impairments. Save the project as CoopCode11 in your working folder.

1. What roles did your team decide are needed to develop the game?
2. How do the tasks necessary to complete the game determine the order in which tasks must be done?

Chapter Summary

Section 11.1 Introduction to Game Design

☐ LO 11.1 Describe the elements of a successful game.

- A basic rule of video game design is that the game should be easy to learn but difficult to master.
- The game-design document details everything about a game, including genre, story, game mechanics, characters, audio, and algorithms.
- Great game stories include a protagonist and antagonist, each with flaws to make them interesting.
- The avatar is the visual representation of the player character.
- Game mechanics describe how the game works.
- The objective of the game provides the reason for playing and what the player must achieve to win.
- Characters include the player character, non-player characters, and, in multiplayer games, other player characters.
- Characters may be friendlies or enemies.

Section 11.2 Finalizing the Game Build

☐ LO 11.2 Explain ways in which to provide game balancing.

- Assets include audio, graphics, videos, and text.
- Some assets become game elements, such as tokens or enemies.
- The game loop is the process of repeatedly launching and processing game elements.
- Game balancing is adjusting the code and gameplay to make the game more fun and playable.
- A game should include skill-building with increasing difficulty in gameplay, but the player must have a reasonable expectation that the game is winnable.

Review and Assessment

Chapter 11 Test

Multiple Choice

Choose the best response for each question.

1. Which type of game generally can be completed in under eight hours?
   A. Side-scroller game
   B. Learning game
   C. Casual game
   D. Genre game
2. How are the genre, story, game mechanics, characters, audio, and algorithms detailed?
   A. The game is created, and then these things are described.
   B. A coder decides what will work and what will not.
   C. A script writer writes the details.
   D. A game-design document is created.
3. How is the player character represented in a game?
   A. As an other player character
   B. As an enemy
   C. By an avatar
   D. By friendly characters
4. Audio, graphics, video, and text are examples of what?
   A. Game elements
   C. Game balancing
   B. Assets
   D. Characters
5. What part of game development is performed to make a game more fun?
   A. Game balancing
   B. Coding
   C. Design document
   D. Algorithm development

Completion

Complete the following sentences with the correct word(s).

6. The _____ is the overall main document that describes the plan for the entire game.
7. A good story has two basic characters, the _____ and the _____, each with flaws to make the character interesting.
8. The _____ of the game is the goal to reach in a game.
9. A basic rule in game balancing is that a game should be _____ to learn, but _____ to master.
10. The player must have a reasonable expectation that the game is _____.

Matching

Match each term with its correct definition.

A. game mechanics
B. game elements
C. assets
D. replayability
E. genre

11. General category to which a game belongs.
12. Characters or objects that are part of the game.
13. Non-code elements of a game.
14. Actions the player can take and how the game reacts to the player.
15. Characteristic of a game that makes a player try again.

Application and Extension of Knowledge

1. Choose a game that you play. Identify one of the game mechanics and analyze it. Write a paragraph describing what coding you think would be needed to make this mechanic possible.
2. Consider the *Sidewalk Scramble* game. Suppose a game mechanic needs to be added to have the player pick up trash along the way and deposit it in the trash bin to score points. Describe how the game design would change.
3. Consider the *Sidewalk Scramble* game. Suppose a game mechanic needs to be added to have multiple lives. Describe how the game design would change.
4. Consider the *Sidewalk Scramble* game. Suppose a game mechanic needs to be added so some game elements appear more often than others. Choose the probability that each game element might appear. Be sure that it adds up to 100 percent. Describe how the game element selection algorithm would change.
5. Drop a tennis ball to see that each bounce is not as high as the one before. This is because energy is depleted with each bounce. In physics, this effect is called decay. Open the final *Sidewalk Scramble* game (HOE11-02C_SidewalkScramble), and save it as AEK11-05 in your working folder. Add a tennis ball game element, and program it to drop from the top of the stage and bounce as it moves to the left. Code decay into the bounce. Each bounce should be shorter than the one before. The player should avoid the tennis ball by using the power of invisibility.

College and Career Readiness

Communication Skills

Reading. Reading for detail involves reading all words and phrases, considering their meaning, and determining how they combine with other elements to convey ideas. Select one chapter of this text. Compare and analyze how generic features are used in each chapter. What did you learn?

Writing. Create a governing game-design document for a game you would like to build, even if it exceeds your current programming skills. Include a backstory, protagonist, antagonist/boss, major story elements, climax, game mechanics, etc. Use a graphic organizer to plan the game and guide your writing of the game-design document. When writing the document, follow standard writing procedures. Use proper grammar and spelling.

Speaking. A societal issue with video games is how violence is represented. Working in small groups, develop a list of regulations that you would like to see developed to control the content in video games. Develop a presentation in which you attempt to persuade your classmates to adopt your regulations. Then, develop a separate presentation in which you attempt to persuade your teacher, acting as a member of the video game industry, to adopt your regulations. How will you alter your presentations for these two different audiences?

Listening. Research the positive and negative aspects of projected future trends in computing. Using the Internet, find video footage of at least three speeches or news broadcasts that discuss this topic. Compare and contrast the speakers' information, points of view, and opinions. How are they similar and different? Using the information presented, create a list of positive and negative aspects of computing.

College and Career Readiness

Portfolio Development

Talents. You have collected documents that show your skills and talents. Select a book report, essay, or poem that you have written that demonstrates your writing talents. If you are an artist, include copies of your completed works. If you are a musician, create a video with segments from your performances.

1. Create a Microsoft Word document that lists your talents. Use the heading "Talents" along with your name. Next to each talent listed, write a description of an assignment or performance and explain how your talent is shown in it. If there is a video, state that it will be made available upon request or identify where it can be viewed online. Indicate that sample screenshots are attached.
2. Scan hard-copy documents related to your talents to serve as samples. Save screenshots from a video, if appropriate, in an appropriate file format. Place hard copies in the container for future reference.
3. Place the video file in an appropriate subfolder for your digital portfolio.
4. Update the master spreadsheet to reflect the inclusion of this file.