Designing the Game: Tank War

- *Homing rockets: Always move in the direction of your opponent*
- *Bouncing bombs: Bounce against walls, and can be used to fire around corners*
- *Shields: Are activated to provide a temporary protective shield*
- *Toolbox: Repairs part of the tank's damage*

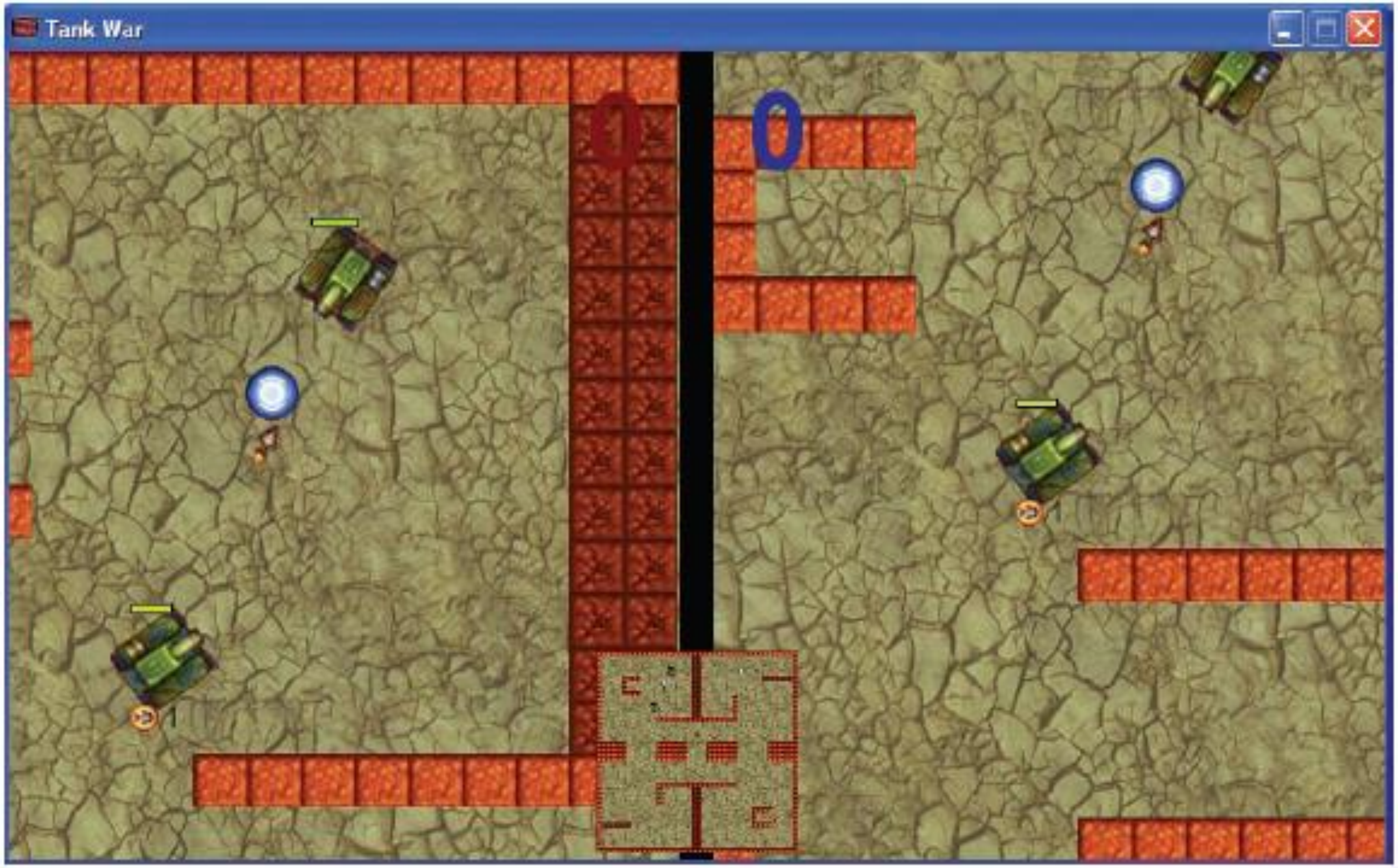


Figure 10-1. *Tank War has a split-screen with a little mini-map at the bottom.*

Playing with Tanks

Creating the arena background and walls:

1. Launch Game Maker and start a new empty game.
2. Create a background resource called `background` using `Background.bmp` from the `Resources/Chapter10` folder on the CD.
3. Create two sprites called `spr_wall1` and `spr_wall2` using `Wall1.gif` and `Wall2.gif`. Disable the **Transparent** property for both sprites.
4. Create a new object called `obj_wall1` and give it the first wall sprite. Enable the **Solid** property and close the object properties. No further behavior is needed.
5. Create a new object called `obj_wall2` and give it the second wall sprite. Enable the **Solid** property and set **Parent** to `obj_wall1`.

Creating the controller object and the room:

1. Create a sound resource called snd_music using Music.mp3 from the Resources/Chapter10 folder on the CD.
2. Create a new object called obj_controller, with no sprite. Set **Depth** to -100 to make sure that the drawing actions we will give it later on are drawn in front of other objects. Add an **Other, Game Start** event and include the **Play Sound** action. Set **Sound** to snd_music and set **Loop** to **true**.
3. Create a new room and switch to the **settings** tab. Call the room room_main and give it an appropriate caption.
4. Switch to the **backgrounds** tab and select the background you created earlier.
5. Switch to the **objects** tab. In the toolbar, set **Snap X** and **Snap Y** to 32, as this is the size of the wall objects.
6. Create a continuous wall of obj_wall1 objects around the edge of the room. Also add walls of both types to the interior so that they create obstacles for the tanks (remember that you can hold the Shift key to add multiple instances of an object).
7. Add one instance of the controller object into the room.

Creating the parent tank object:

1. Create a new object called obj_tank_parent, with no sprite.
2. Add a **Create** event and include a **Set Friction** action with **Friction** set to 0.5. This will cause the tanks to naturally slow down and come to rest when the player is not pressing the acceleration key.
3. Add a **Collision** event with obj_wall1 and include a **Set Variable** action. Set **Variable** to speed and **Value** to -speed. This will reverse the tank's movement direction when it collides with a wall.
4. Likewise, add a **Collision** event with obj_tank_parent and include a **Set Variable** action. Set **Variable** to speed and **Value** to -speed (you could also right-click on the previous collision event and select **Duplicate Event** to achieve this).

Creating the two players' tank objects:

1. Create two sprites called spr_tank1 and spr_tank2 using Tank1.gif and Tank2.gif. Set the **Origin** of both sprites to **Center**. Note that these sprites have 60 subimages corresponding to different facing directions for the tanks.

2. Create a new object called obj_tank1 and give it the first tank sprite. Set **Parent** to obj_tank_parent and enable the **Solid** option. Set **Depth** to -5 to make sure it appears in front of other objects, such as shells, later on.

3. Add a **Keyboard, Letters, A** event and include a **Set Variable** action. Set **Variable** to direction and **Value** to 6, and enable the **Relative** option. This will rotate the tank anticlockwise.

4. Add a **Keyboard, Letters, D** event and include a **Set Variable** action. Set **Variable** to direction and **Value** to -6, and enable the **Relative** option. This will rotate the tank clockwise.

5. Add a **Keyboard, Letters, W** event and include a **Test Variable** action. Set **Variable** to speed, **Value** to 8, and **Operation** to **smaller than**. Include a **Set Variable** action, setting **Variable** to speed and **Value** to 1 and enabling the **Relative** option. This will then only increase the speed if it is smaller than 8.

6. Add a **Keyboard, Letters, S** event and include a **Test Variable** action. Set **Variable** to speed, **Value** to -8, and **Operation** to **larger than**. Include a **Set Variable** action, setting **Variable** to speed and **Value** to -1 and enabling the **Relative** option. This will only reduce the speed (reverse) if the speed is greater than -8 (full speed backward).

7. Add a Step, End Step event. In this event we must set the subimage of the sprite that corresponds to the direction the tank is facing. Include the Change Sprite action, setting Sprite to `spr_tank1`, Subimage to `direction/6` and Speed to `0`. As in Galactic Mail, `direction/6` converts the angle the object is facing (between 0 and 360) to the range of images in the sprite (between 0 and 60).

8. We will draw the tank ourselves because later we want to draw more than just the sprite. Add a Draw event. Include the Draw Sprite action, setting Sprite to `spr_tank1` and Subimage to `-1` and enabling the Relative option.

9. Repeat steps 2–8 (or duplicate `obj_tank1` and edit it) to create `obj_tank2`. This time you should use the arrow key events to control its movement (Keyboard, Left, etc.)

10. Reopen the room and put one instance of each tank into it.

Firing Shells

Recording the player's score in the controller object:

1. Create a font called fnt_score and select a font like Arial with a **Size** of 48 and the **Bold** option enabled. We only need to use the numerical digits for the score, so you can click the **Digits** button to leave out the other characters in the font. This will save storage space and reduce the size of your .gm6 and executable game files.
2. Reopen the controller object and select the **Game Start** event. Include a **Set Variable** action with **Variable** set to global.score1 and **Value** set to 0. Include another **Set Variable** action with **Variable** set to global.score2 and **Value** also set to 0. This creates and initializes the global score variables that will store the player's score.
3. Add a **Draw** event and include a **Set Font** action. Set **Font** to fnt_score and **Align** to **right**. Include a **Set Color** action and choose a dark red color.
4. Include a **Draw Variable** action from the **control** tab. Set **Variable** to global.score1, **X** to 300, and **Y** to 10.
5. Include another **Set Font** action with **Font** set to fnt_score, but this time set **Align** to **left**. Include a **Set Color** action and choose a dark blue color.
6. Include a **Draw Variable** action with **Variable** set to global.score2, **X** set to 340, and **Y** set to 10.

Creating the large explosion object:

1. Create a sprite called `spr_explosion_large` using `Explosion_large.gif` and **Center** the **Origin**.
2. Create a sound called `snd_explosion_large` using `Explosion_large.wav`.
3. Create a new object called `obj_explosion_large`. Give it the large explosion sprite and set **Depth** to `-10`. Add a **Create** event and include a **Play Sound** action, with **Sound** set to `snd_explosion_large` and **Loop** set to **false**.
4. Add an **Other, Animation End** event and include the **Restart Room** action.

Creating the small explosion object:

1. Create a sprite called `spr_explosion_small` using `Explosion_small.gif` and **Center** the **Origin**.
2. Create a sound called `snd_explosion_small` using the file `Explosion_small.wav`.
3. Create an object called `obj_explosion_small`. Give it the small explosion sprite and set **Depth** to `-10`. Add a **Create** event and include the **Play Sound** action, with **Sound** set to `snd_explosion_small` and **Loop** set to **false**.
4. Add the **Other, Animation End** event and include the **Destroy Instance** action.

Adding a damage mechanism to the parent tank object:

1. Reopen obj_tank_parent and select the **Create** event. Include a **Set Variable** action with **Variable** set to damage and **Value** set to 0.
2. Add a **Step, Step** event and include a **Test Variable** action. Set **Variable** to damage, **Value** to 100, and **Operation** to **smaller than.** Include an **Exit Event** action so that no further actions are executed if the damage is smaller than 100.
3. Now we need to find out what type of tank we are dealing with. Include a **Test Variable** action with **Variable** set to object_index, **Value** set to obj_tank1, and **Operation** set to **equal to.** Include a **Set Variable** action with **Variable** set to global.score2, **Value** set to 1, and the **Relative** option enabled. This will then increase player two's score if this instance is player one's tank.
4. Include an **Else** action followed by a **Set Variable** action. Set **Variable** to global.score1 and **Value** to 1, and enable the **Relative** option. This will increase player one's score if this instance is player two's tank.
5. Include a **Create Instance** action with **Object** set to obj_explosion_large and the **Relative** option enabled.
6. Finally, include a **Destroy Instance** action.

Adding a draw event to the parent tank object to draw the health bars:

1. Add a **Draw** event for the parent tank object.
2. Include a **Set Health** action (**score** tab) and set **Value** to `100-damage`. Damage is the opposite concept to health, so subtracting it from 100 makes this conversion (e.g., 80 percent damage converts to 100 – 80 = 20 percent health).
3. Add a **Draw Health** action. Set **X1** to `-20`, **Y1** to `-35`, **X2** to `20`, and **Y2** to `-30`. Enable the **Relative** option, but leave the other parameters as they are. This will draw a small health bar above the tank. It may seem strange to be using the health functions here as they only work with one health value and we have two players. However, this technique works because we set the health in step 2 using the instance's own `damage` variable, just before we draw the health bar.
4. Reopen `obj_tank1` and select the **Draw** event. Include the **Call Parent Event** action (**control** tab) at the end of the list of actions for this event. This will make sure that the **Draw** event of the parent tank object is also executed.
5. Reopen `obj_tank2` and select the **Draw** event. Include the **Call Parent Event** action (**control** tab) at the end of the list of actions for this event.

Creating the parent shell object:

1. Create a sprite called `spr_shell` using `Shell.gif` and **Center** the **Origin**. Note that like the tank sprite, this contains 60 images showing the shell pointing in different directions.

2. Create a new object called `obj_shell_parent` and leave it without a sprite (you can set it, but it isn't necessary for the parent as it never appears in the game).

3. Add a **Create** event and include the **Set Alarm** action. Set the **Number of Steps** to `30` and select **Alarm 0**.

4. Add an **Alarm, Alarm 0** event and include the **Destroy Instance** action.

5. Add a **Step, End Step** event and include the **Change Sprite** action. Set **Sprite** to `spr_shell`, **Subimage** to `direction/6`, and **Speed** to `0` (to stop it from animating).

6. Add a **Collision** event with `obj_wall1` and include a **Create Instance** action. Set **Object** to `obj_explosion_small` and enable the **Relative** option. Also include a **Destroy Instance** action to destroy the shell.

7. Add a **Collision** event with `obj_wall2`. This object must be temporarily removed. Include a **Create Instance** action with **Object** set to `obj_explosion_small` and the **Relative** option enabled. Include a **Jump to Position** action with **X** and **Y** set to `100000`. Also select the **Other** object for **Applies to** so that the wall is moved rather than the shell.

8. Include a **Set Alarm** action and select the **Other** object for **Applies to** so that it sets an alarm for the wall. Select **Alarm 0** and set **Number of Steps** to `300`. Finally, include a **Destroy Instance** action to destroy the shell.

9. Add a **Collision** event with `obj_shell_parent` and include a **Create Instance** action. Set **Object** to `obj_explosion_small` and enable the **Relative** option. Also include a **Destroy Instance** action to destroy the shell.

Editing the destructible wall object to make it reappear:

1. Reopen the `obj_wall2` object and add an **Alarm, Alarm 0** event. Include a **Check Empty** action with **X** set to `xstart`, **Y** set to `ystart`, and **Objects** set to **All**. Include a **Jump to Start** action.

2. Next include an **Else** action followed by a **Set Alarm** action. Select **Alarm 0** and set **Number of Steps** to `5`. That way, when the position is not empty it will wait five more steps and then try again.

Creating the players' shell objects:

1. Create a new object called obj_shell1. Give it the shell sprite and set its **Parent** to obj_shell_parent.
2. Add a **Collision** event with obj_tank2 and include a **Set Variable** action. Set **Variable** to damage and **Value** to 10, and enable the **Relative** option. Also select the **Other** object for **Applies to** so that the tank's damage variable is changed.
3. Include a **Create Instance** action with **Object** set to obj_explosion_small and enable the **Relative** option. Also include a **Destroy Instance** action to destroy the shell.
4. Repeat steps 1–3 to create obj_shell2 using a **Collision** event with obj_tank1 rather than obj_tank2.

Adding events to make the tank objects fire shells:

1. Reopen the parent tank object and select the **Create** event. Include a **Set Variable** action with **Variable** set to can_shoot and **Value** set to 0.

2. Select the **Step** event and include a **Set Variable** action at the beginning of the list of actions. Set **Variable** to can_shoot and **Value** to 1, and enable the **Relative** option.

3. Reopen obj_tank1 and add a **Key Press, <Space>** event. Include the **Test Variable** action, with **Variable** set to can_shoot, **Value** set to 0, and **Operation** set to **smaller than**. Next include the **Exit Event** action so that the remaining actions are only executed when can_shoot is larger or equal to 0.

4. Include a **Create Moving** action. Set **Object** to obj_shell1, **Speed** to 16, and **Direction** to direction, and enable the **Relative** option. Also include a **Set Variable** action with **Variable** set to can_shoot and **Value** set to -10.

5. Repeat steps 3–4 for the obj_tank2, this time using a **Key Press, <Enter>** event for the key and obj_shell2 for the **Create Moving** action.

Secondary Weapons

Creating the pickup object:

1. Create a sprite called spr_pickup using Pickup.gif. Note that it consists of four completely different subimages, representing each different kind of pickup.
2. Create a new object called obj_pickup and give it the pickup sprite.
3. Add a **Create** event and include the **Set Variable** action. Set **Variable** to kind and **Value** to choose(0,1,2,3). This will choose randomly between the numbers in brackets that are separated by commas.
4. Include the **Set Alarm** action for **Alarm 0** and set **Number of Steps** to 100+random(500). This will give a random time between 100 and 600 steps or about 3 and 20 seconds. Finally, include a **Jump to Random** action with the default parameters. This will move the instance to a random empty position.
5. Add an **Alarm, Alarm 0** event and include the **Set Variable** action. Set **Variable** to kind and **Value** to choose(0,1,2,3).
6. Include the **Set Alarm** action for **Alarm 0** with **Number of Steps** set to 100+random(500). Finally, include a **Jump to Random** action.
7. Add a **Collision** event with obj_tank_parent and include a **Jump to Random** action.
8. Add a **Draw** event and include the **Draw Sprite** action. Set **Sprite** to spr_pickup, **Subimage** to kind and enable the **Relative** option.

Editing the parent tank object to record pickups:

1. Reopen obj_tank_parent and select the **Create** event.
2. Include a **Set Variable** action with **Variable** set to weapon and **Value** set to -1. Include a second **Set Variable** action with **Variable** set to ammunition and **Value** set to 0.
3. Add a **Collision** event with obj_pickup and include a **Test Variable** action. Set **Variable** to other.kind, **Value** to 3, and **Operation** to **equal to**. A value of 3 corresponds to the toolbox. This needs to repair the tank's damage, so include a **Start Block** action to begin the block of actions that do this.
4. Include a **Set Variable** action with **Variable** set to weapon and **Value** set to -1. Include a second **Set Variable** action with **Variable** set to damage and **Value** set to max(0,damage-50). The function max decides which is the largest of the two values you give it (more about functions in Chapter 12). Therefore, this sets the new damage to the largest out of damage-50 and 0. In effect, this subtracts 50 from damage but makes sure it does not become smaller than 0. Include an **End Block** action.
5. Include an **Else** action, followed by a **Start Block** action to group the actions that are used if this is not a toolbox pickup.
6. Include a **Set Variable** action with **Variable** set to weapon and **Value** set to other.kind. Include another **Set Variable** action with **Variable** set to ammunition and **Value** set to 10.
7. Finally, include an **End Block** action.

Displaying the secondary weapon in the parent tank object:

1. Create a new sprite called spr_weapon using Weapon.gif. Note that it consists of three subimages (no image is required for the toolbox).
2. Create a font called fnt_ammunition and keep the default settings for it.
3. Select the **Draw** event in obj_tank_parent and include a **Test Variable** action. Set **Variable** to weapon, **Value** to -1, and **Operation** to **larger than**. This will ensure that we only draw something when there is a secondary weapon. Include a **Start Block** action to group the drawing actions.
4. Include the **Draw Sprite** action and select spr_weapon. Set **X** to -20, **Y** to 25, and **Subimage** to weapon. Also enable the **Relative** option.
5. Include a **Set Color** action and choose black. Then include a **Set Font** action, selecting fnt_ammunition and setting **Align** to **left**.
6. Next include a **Draw Variable** action with **Variable** set to ammunition, **X** set to 0, **Y** set to 24, and the **Relative** option enabled.
7. Finally, include an **End Block** action to conclude the actions that draw the weapon information.

Creating the parent rocket object:

1. Create a sprite called spr_rocket using Rocket.gif and **Center** the **Origin**.
2. Create a new object called obj_rocket_parent and set **Parent** to obj_shell_parent.
3. Add a **Create** event and include the **Set Alarm** action. Set **Number of Steps** to 60 and select **Alarm 0**.
4. Add a **Step, End Step** event and include a **Change Sprite** action. Select the rocket sprite, then set **Subimage** to direction/6 and **Speed** to 0.

Next we create the two actual rocket objects.

Creating the actual rocket objects:

1. Create a new object called it obj_rocket1 and give it the rocket sprite. Set **Parent** to obj_rocket_parent.
2. Add a **Create** event and include the **Move Towards** action. Set **X** to obj_tank2.x, **Y** to obj_tank2.y, and **Speed** to 8.
3. Add a **Collision** event with obj_tank2 and include a **Set Variable** action. Select **Other** from **Applies to** (the tank), set **Variable** to damage and **Value** to 10, and enable the **Relative** option.
4. Include a **Create Instance** action, selecting obj_explosion_small and enabling the **Relative** option. Also include a **Destroy Instance** action.
5. Create obj_rocket2 in the same way, but move toward obj_tank1 in the **Create** event, and add a **Collision** event with obj_tank1 for the actions in steps 3 and 4.

Adding events to shoot rockets for the tank object:

1. Reopen the first tank object and add a **Key Press, <Ctrl>** event. Include a **Test Variable** action, with **Variable** set to `can_shoot`, **Value** set to `0`, and **Operation** set to **smaller than.** Next include the **Exit Event** action so that the remaining actions are only executed when `can_shoot` is larger than or equal to 0.

2. Include the **Test Variable** action, with **Variable** set to `weapon`, **Value** set to `0`, and **Operation** set to **equal to.** Next include a **Test Instance Count** action with **Object** set to `obj_tank2`, **Number** set to `0` and **Operation** set to **larger than.** Follow this with a **Create Instance** action for `obj_rocket1`, and enable the **Relative** option. This creates a rocket only when it is the current secondary weapon and the other tank exists (this avoids a rare error when the other tank has just been destroyed).

3. Next we need to decrease the ammunition. Include a **Set Variable** action with **Variable** set to `ammunition`, **Value** set to `-1`, and the **Relative** option enabled. Include a **Test Variable** action with **Variable** set to `ammunition`, **Value** set to `1`, and **Operation** set to **smaller than.** Follow this with a **Set Variable** action with **Variable** set to `weapon` and **Value** set to `-1`.

4. Finally, include a **Set Variable** action with **Variable** set to `can_shoot` and **Value** set to `-10`.

5. Repeat steps 1–4 for `obj_tank2`, using a **Key Press, Others, <Delete>** event and creating `obj_rocket2`.

Creating the bouncing bomb objects:

1. Create a sprite called spr_bouncing using Bouncing.gif and **Center** the **Origin**.
2. Create a new object called obj_bouncing_parent and set its **Parent** to obj_shell_parent.
3. Add a **Collision** event with obj_wall1 and include the **Bounce** action. Select **precisely** and set **Against** to **solid objects**.
4. Add a similar **Collision** event with obj_wall2.
5. Add a **Step, End Step** event and include a **Change Sprite** action. Select spr_bouncing, set **Subimage** to direction/6, and set **Speed** to 0.
6. Create a new object called obj_bouncing1 and give it the bouncing bomb sprite. Set its **Parent** to obj_bouncing_parent.
7. Add a **Collision** event with obj_tank2 and include a **Set Variable** action. Select **Other** from **Applies to**, set **Variable** to damage, set **Value** to 10, and enable the **Relative** option. Include a **Create Instance** action for obj_explosion_small and enable the **Relative** option.
8. Include a **Destroy Instance** action.
9. Repeat steps 6 and 7 to create obj_bouncing2 using a **Collision** event with obj_tank1.

Editing the parent tank object to support shields:

1. Create sprites called spr_shield1 and spr_shield2 using Shield1.gif and Shield2.gif and **Center** their **Origins**.
2. Reopen the parent tank object and select the **Create** event. Include a **Set Variable** action with **Variable** set to shield and **Value** set to 0.
3. Select the **Step** event and include a **Set Variable** action at the start of the list. Set **Variable** to shield, set **Value** to -1, and enable the **Relative** option.
4. Reopen obj_shell1 and select the **Collision** event with obj_tank2. Include a **Test Variable** action directly above the **Set Variable** that increases the damage. Select **Other** from **Applies to**, then set **Variable** to shield, **Value** to 0, and **Operation** to **smaller than**. Now the damage will only be increased when the tank has no shield.
5. Repeat step 4 for objects obj_shell2, obj_rocket1, obj_rocket2, obj_bouncing1, and obj_bounding2.
6. Reopen obj_tank1 and select the **Draw** event. Include a **Test Variable** action at the start of the action list. Set **Variable** to shield, **Value** to 0, and **Operation** to **larger than**. Follow this with a **Draw Sprite** action for spr_shield1 with the **Relative** option enabled.
7. Repeat step 6 for obj_tank2, this time drawing spr_shield2.

Editing tank objects to shoot bombs and use shields:

1. Reopen obj_tank1 and select the **Key Press, <Ctrl>** event.
2. Include a **Test Variable** action below the **Create Instance** action that creates obj_rocket1. Set **Variable** to weapon, **Value** to 1, and **Operation** to **equal to**. Follow this with a **Create Moving** action for obj_bouncing1, setting **Speed** to 16 and **Direction** to direction, and enabling the **Relative** option.
3. Include another **Test Variable** action below this, with **Variable** set to weapon, **Value** set to 2, and **Operation** set to **equal to**. Follow this with by a **Set Variable** action with **Variable** set to shield and **Value** set to 40.
4. Repeat steps 1–3 for obj_tank2, adapting the **Key Press, <Delete>** event and creating obj_bouncing2.

Views

- **View in room**: This is an area of the room that needs to be displayed in the view. The **X** and **Y** positions define the top-left corner of this area and **W** and **H** specify the width and height of it.
- **Port on screen**: This is the position on the window where the view should be shown. The **X** and **Y** positions define the top-left corner of this area and **W** and **H** specify the width and height of it. If the width and height are different from the size of the view area, then the view will be automatically scaled to fit. Game Maker will also automatically adapt the size of the window so that all ports fit into it.
- **Object following**: Specifying an object here will make the view track that object as it moves around the room. **Hbor** and **Vbor** specify the size of the horizontal and vertical borders that you want to keep around the object. The view will not move until the edge of the screen is closer than this distance from the object. Setting **Hbor** to half the width of the view and **Vbor** to half the height of the view will therefore maintain the object in the center. Finally, **Hsp** and **Vsp** allow you to limit the speed with which the view moves (–1 means no limit).

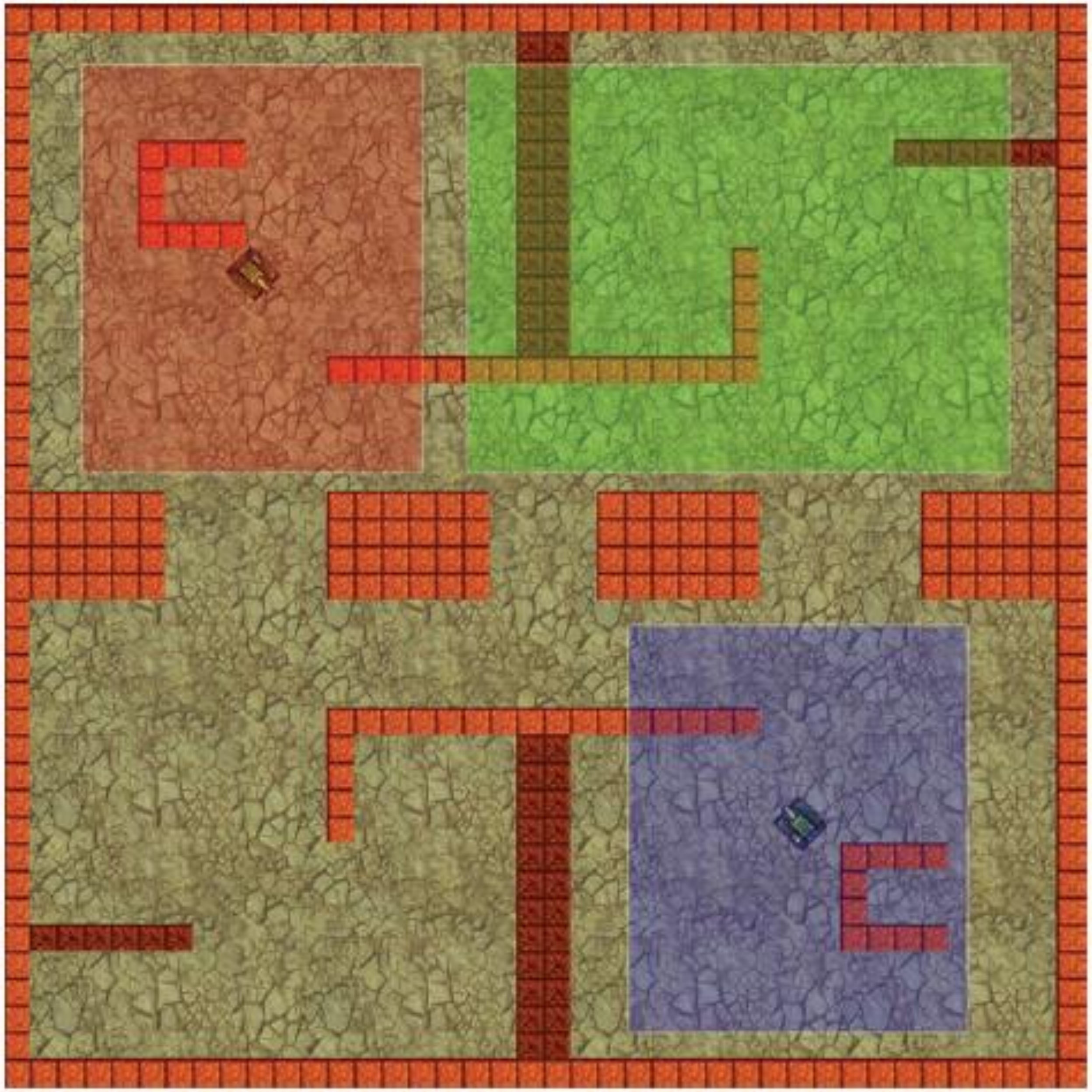
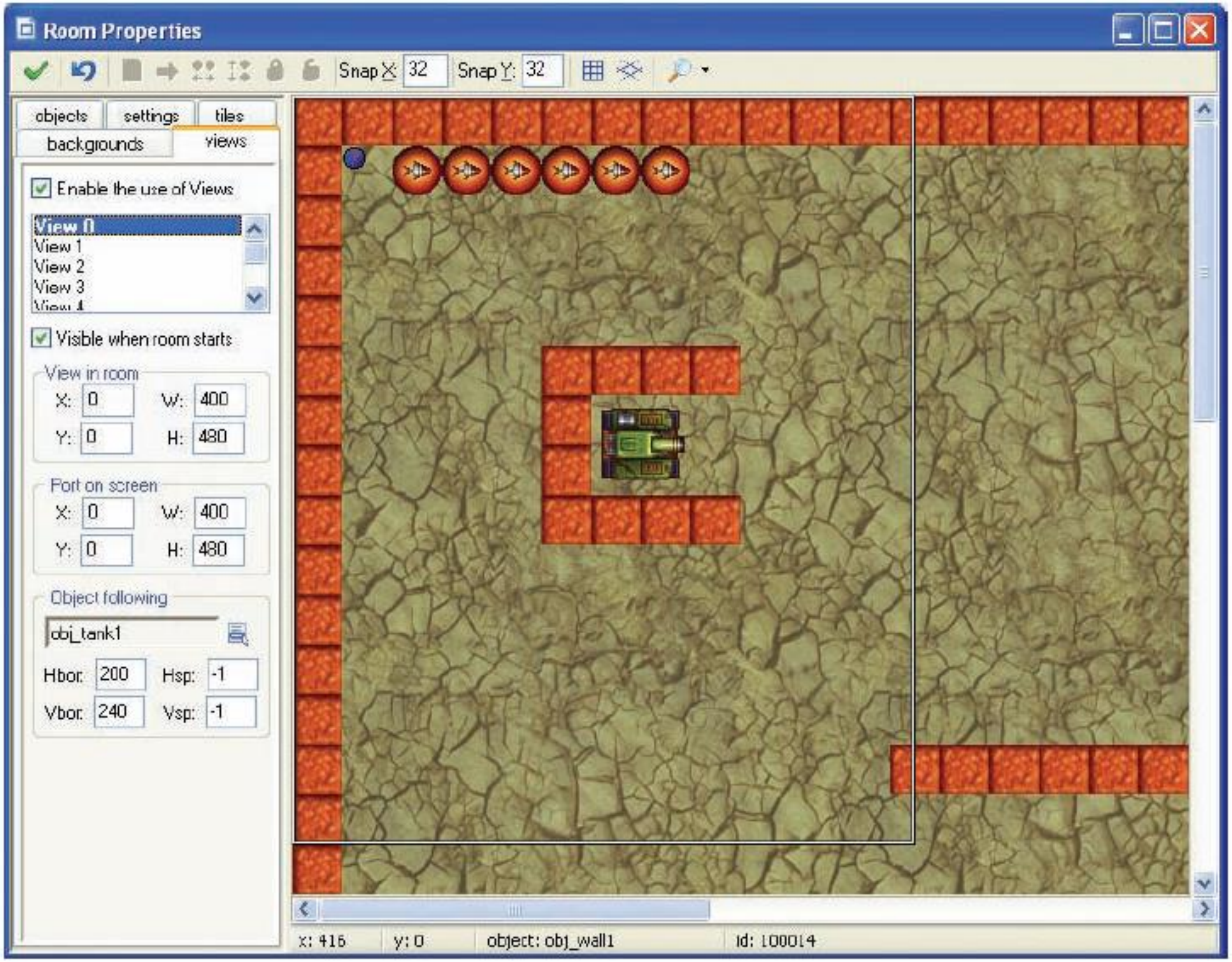


Figure 10-2. *We'll create a large room, much bigger than a normal window (green rectangle), and provide views into it for each of the tanks (red and blue squares).*

Editing the room resource to provide two views:

1. Reopen the main room and switch to the **settings** tab.
2. Set both the **Width** and **Height** of the room to 1280, to create a much larger room.
3. Switch to the **objects** tab and add wall instances to incorporate the extra playing area. Start the tanks close to two opposite corners and add six pickup instances. Also don't forget that the room needs exactly one instance of the controller object.
4. Switch to the **views** tab and select the **Enable the use of Views** option. This activates the use of views in this room.
5. Make sure that **View 0** is selected in the list and enable the **Visible when room starts** option. We will use this view for player one.
6. Under **View in room** set **X** to 0, Y to 0, W to 400, and H to 480. The **X** and **Y** positions of the views don't really matter in this case as we will make them follow the tanks. Nonetheless, notice that lines appear in the room to indicate the size and position of the view.
7. Under **Port on screen** set **X** to 0, **Y** to 0, **W** to 400, and **H** to 480. This port will show player one's view on the left side of the screen.
8. Under **Object following** select `obj_tank1`, then set **Hbor** to 200 and **Vbor** to 240. The form should now look like Figure 10-3.

9. Now select **View 1** in the list and enable the **Visible when room starts** option. We will use this view for player two.
10. Under **View in room** set **X** to `0`, **Y** to `0`, **W** to `400`, and **H** to `480`.
11. Under **Port on screen** set **X** to `420`, **Y** to `0`, W to `400`, and H to `480`. This places the second view to the right of the first view with a little space between them.
12. Under **Object following** select `obj_tank2`, and set **Hbor** to `200` and **Vbor** to `240`.



Enable the use of Views ✓

Visible when room starts ✓

View in room

X:	0	W:	400
Y:	0	H:	480

Port on screen

X:	0	W:	400
Y:	0	H:	480

Object following: obj_tank1

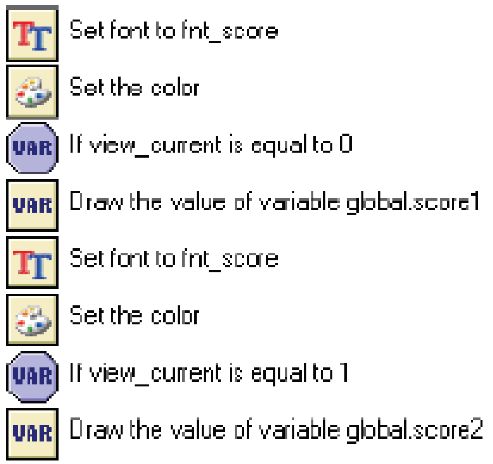
Hbor:	200	Hsp:	-1
Vbor:	240	Vsp:	-1

Figure 10-3. *This is how the form should look when the values for View 0 have been set.*

Tip The empty region between the views defaults to the color black. You can change this in the **Global Game Settings** on the **graphics** tab under **Color outside the room region**.

Editing the controller object to draw the score relative to the view position:

1. Reopen the controller object and select the **Draw** event.
2. Include a **Test Variable** action before the **Draw Variable** action that draws the score for player one. Set **Variable** to `view_current`, **Value** to `0`, and **Operation** to **equal to**.
3. Edit the **Draw Variable** action that draws player one's score. Change **X** to `view_xview[0]+380` and **Y** to `view_yview[0]+10`.
4. Include a **Test Variable** action before the **Draw Variable** action that draws the score for player two. Set **Variable** to `view_current`, **Value** to `1`, and **Operation** to **equal to**.
5. Edit the **Draw Variable** action for player two. Change **X** to `view_xview[1]+20` and **Y** to `view_yview[1]+10`. The action list should now look like Figure 10-4.



Set font to fnt_score
Set the color
If view_current is equal to 0
Draw the value of variable global.score1
Set font to fnt_score
Set the color
If view_current is equal to 1
Draw the value of variable global.score2

Figure 10-4. *These actions draw the scores correctly for each view.*

Adding a view to create a mini-map:

1. Reopen the main room and switch to the **views** tab.
2. Select **View 2** in the list and enable the **Visible when room starts** option.
3. Under **View in room** set X to `0`, Y to `0`, W to `1280`, and H to `1280` (the entire room).
4. Under **Port on screen** set X to `350`, Y to `355`, W to `120`, and H to `120`. No object needs to be followed.

Tip To improve the mini-map and make it more "iconic," you could make the different objects draw something different when the variable `view_current` is equal to 2. For example, the pickup object could simply display a red disk and the walls could draw black squares.