Source Control with Cracker Chase

Collaborating on a code base using Git

Cracker Chase is a small game built using the MonoGame framework. It is not very good as a game, and this implementation has a number of flaws.

You task for this lab exercise is to work collaboratively as a team to make some improvements to the code base and the game.

The first part of these instructions will guide you through some git commands and the second part will let you loose on a number of desired improvements and it will be up to you to work together to get it done.

In order to do this you will each need a GitHub account. If you don't yet have one, please register. It is free.

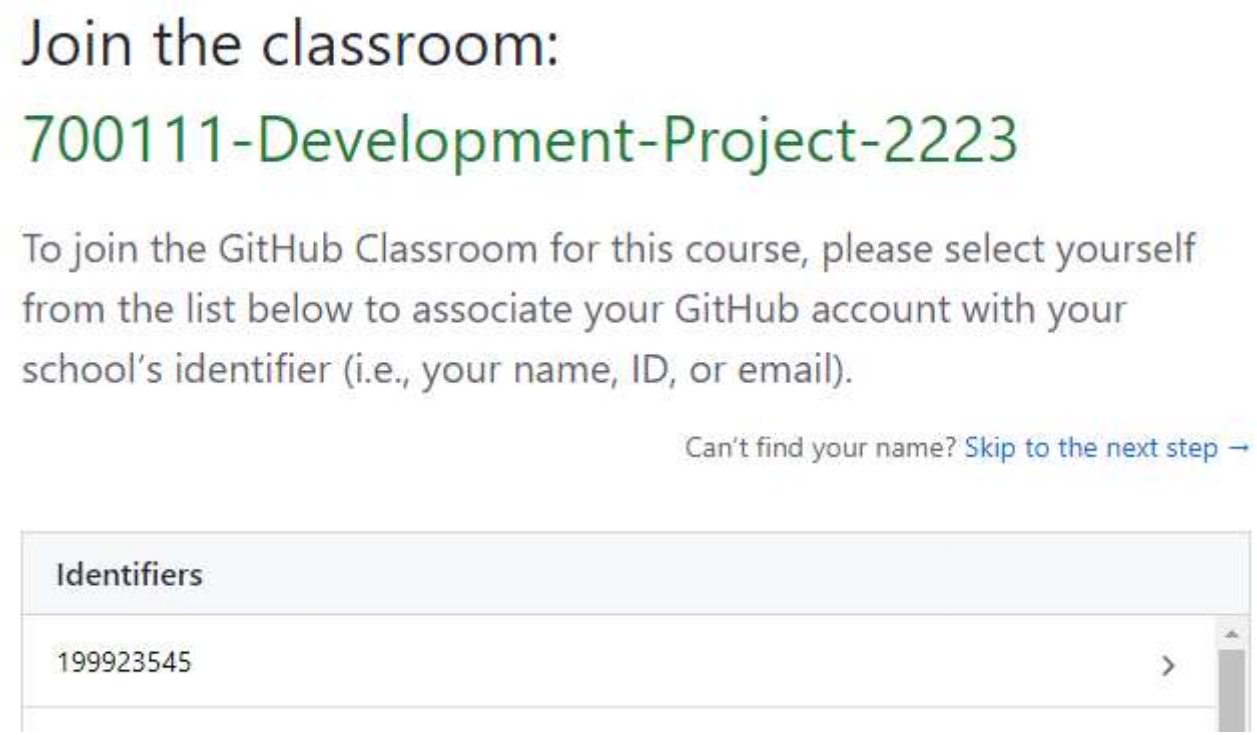
You will also want to have a Git client installed. The instructions assume that you have command line tools installed such as these: **https://git-scm.com/download/win (https://git-scm.com/download/win)**

GitHub Classroom

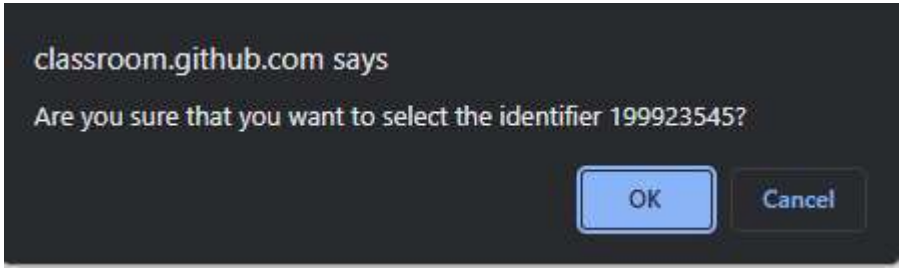
Next up we are going to connect to the GitHub Classroom Team repository. You start connecting by clicking on the following link:

https://classroom.github.com/a/1Z8l6Bpv (https://classroom.github.com/a/1Z8l6Bpv)

That should take you through to this page. You will need to scroll down the list to find your 9 digit student number. Click on it.



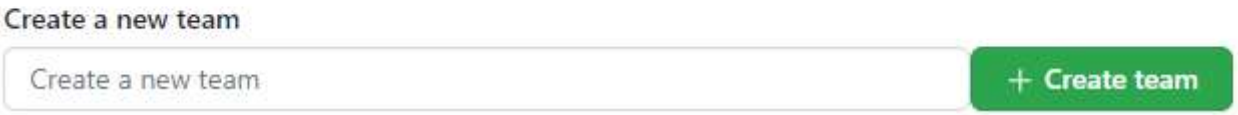
There will be a pop up to confirm that you have chosen the correct id.



Next you will be asked which team you belong to. If you are the first in your team to join, then you will need to type in the team name. The team name will always be an animal name plus a code for the academic year. For example: "antelope 2223". Otherwise you should be able to select it from the list of existing teams.

Accept the group assignment — 700111-Development-Project-2223-CrackerChase

Before you can accept this assignment, you must create or join a team. Be sure to select the correct team as you won't be able to change this later.



Now that you are part of a team, you can join the assignment. Click the "Accept this assignment" button.

700111-Development-Project-2223

Accept the assignment — 700111-Development-Project-2223-CrackerChase

Once you accept this assignment, you will be granted access to the `700111-development-project-2223-crackerchase-DavidParkerDr` repository in the 700111-Development-Project organization on GitHub.

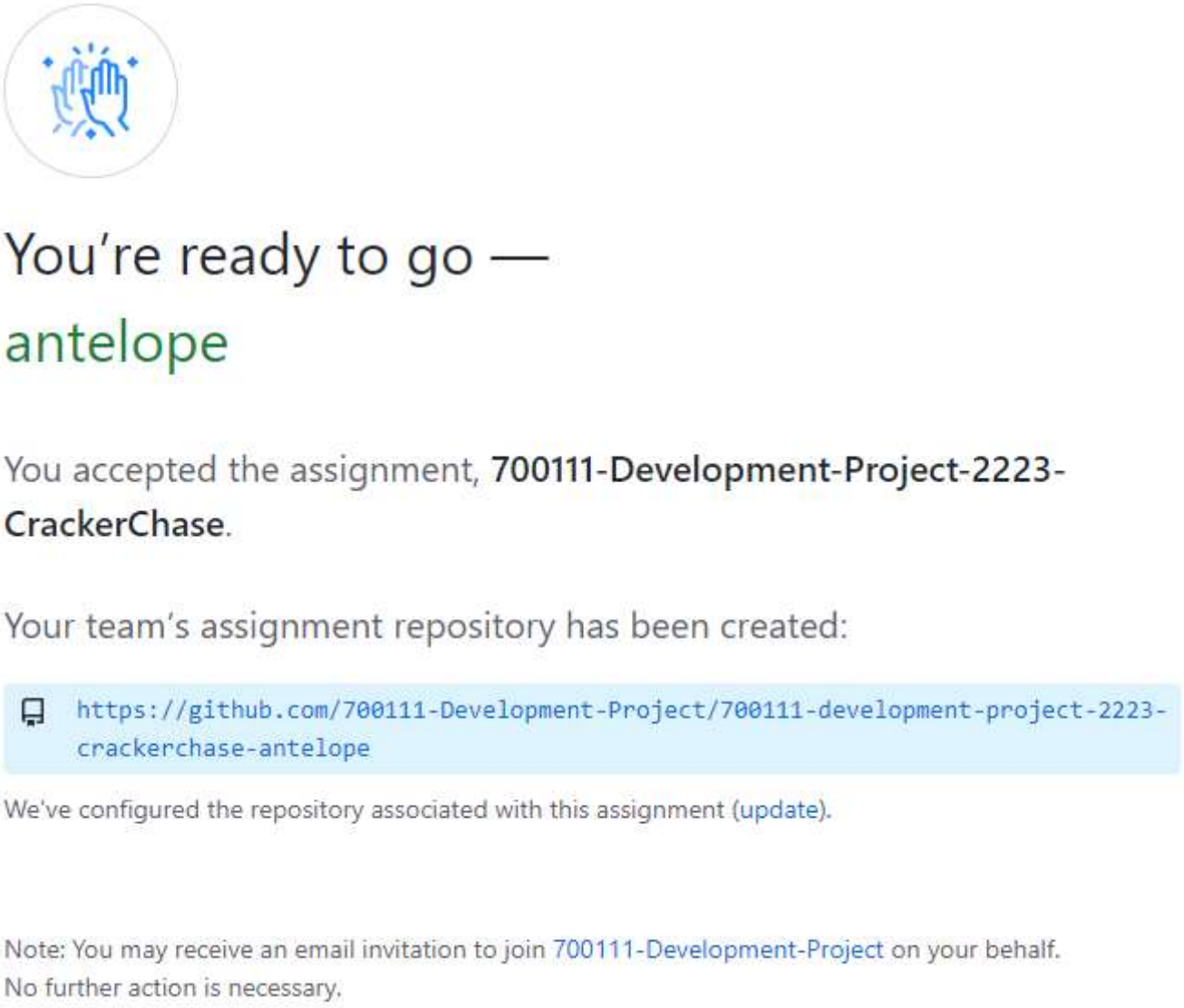
Accept this assignment

There will be a confirmation that the repository is being set up. If you refresh the page then it should be ready.

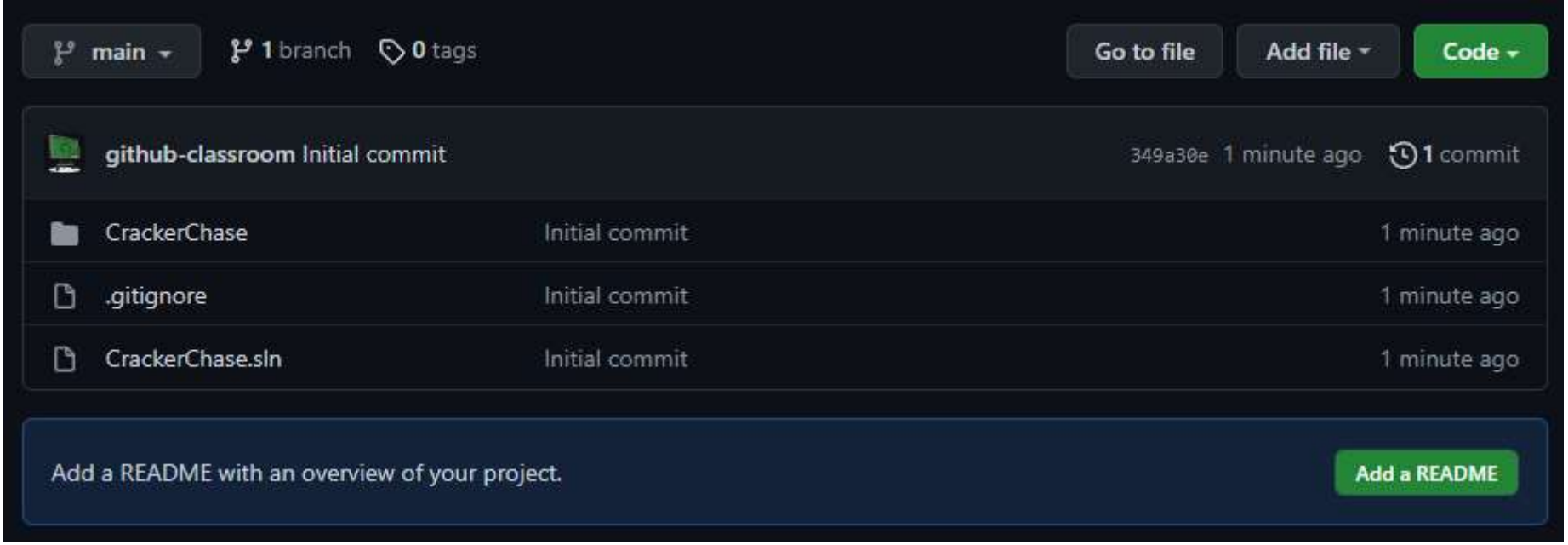
You accepted the assignment, **700111-Development-Project-2223-CrackerChase** . We're configuring your repository now. This may take a few minutes to complete. Refresh this page to see updates.

Note: You may receive an email invitation to join 700111-Development-Project on your behalf.
No further action is necessary.

Something like this where "antelope" will be replaced with whatever your team name is.

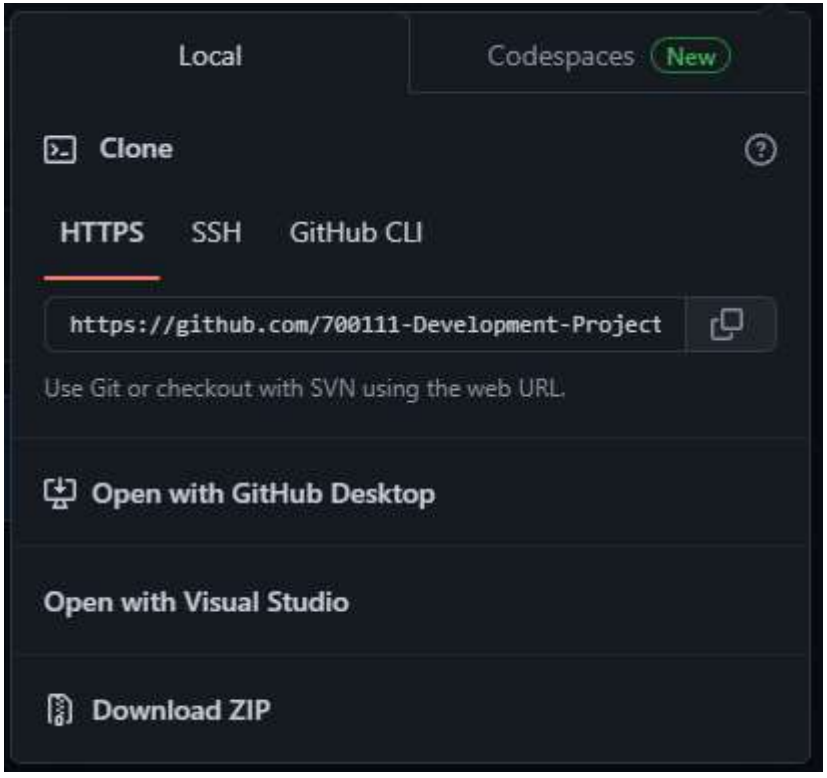


You can now click on the GithHub link to access your repository.



Cloning the repository

Now that we have access to the repository we are going to clone it. Click on the green "Code" button. This will reveal a panel with a url in it. You can click on the clipboard button which should copy the url ready for pasting shortly.



Right, now we need somewhere for the local repository to go. You probably want this to be on your G: drive, but you can choose.

Create an empty directory.

Now open a command prompt. Navigate to the folder that you created in the command prompt.

Type “git clone “ and then paste the git url that you copied earlier. Press the enter key to execute.

You should see something like the following:

```
E:\Student GitHubs>git clone https://github.com/700111-Development-Project/700111-development-project-
antelope.git
Cloning into '700111-development-project-2223-crackerchase-antelope'...
remote: Enumerating objects: 22, done.
remote: Counting objects: 100% (22/22), done.
remote: Compressing objects: 100% (19/19), done.
remote: Total 22 (delta 3), reused 20 (delta 3), pack-reused 0R
Receiving objects: 100% (22/22), 1.36 MiB | 13.70 MiB/s, done.
Resolving deltas: 100% (3/3), done.

E:\Student GitHubs>
```

You can change directory to the folder that was just created by typing "cd " plus the name of the newly created directory. If you type “dir” and execute, you should see that the contents of the repository should now be in the folder.

```
E:\Student GitHubs>cd 700111-development-project-2223-crackerchase-antelope

E:\Student GitHubs\700111-development-project-2223-crackerchase-antelope>dir
 Volume in drive E is QVO
 Volume Serial Number is F4FB-D65D

 Directory of E:\Student GitHubs\700111-development-project-2223-crackerchase-antelope

18/01/2023  15:50    <DIR>          .
18/01/2023  15:50    <DIR>          ..
18/01/2023  15:50               468 .gitignore
18/01/2023  15:50    <DIR>          CrackerChase
18/01/2023  15:50             1,141 CrackerChase.sln
               2 File(s)          1,609 bytes
               3 Dir(s)  431,529,299,968 bytes free
```

In Windows Explorer, if view hidden files is selected, then you will also see a .git folder. This folder, in the root of the cloned folder, holds all of the bookkeeping files for operating the local git repository.

In the command prompt, type “git log”.

This will show a list of all the previous commits and messages along with information about who made the commits.

Notice the coloured text. This is an indication of which parts of the repository this is in sync with.

origin/main is the remote repository, main branch.

origin/HEAD indicates that it is the latest version on the remote repository.

HEAD -> main indicates that it is the latest version in the local repository too.

Note: earlier repositories in GitHub defaulted to "master" rather the "main". Some of the screenshots or text may say "master", but you can just replace it with "main" instead.

```
E:\Student GitHubs\700111\crackerTest>git log
commit 349a30eb56230e5c8be41adff11a488e59c95bbc (HEAD -> main, origin/main, origin/HEAD)
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date:   Fri Jan 21 11:46:39 2022 +0000

    Initial commit

E:\Student GitHubs\700111\crackerTest>
```

In the base CrackerChase solution there are a number of cs files that were hastily created and have a bunch of using statements that are surplus to requirements. It would be handy if we could tidy up a bit and remove all the ones that aren’t being used.

Game1, Sprite, Target, and Mover files all need attending to so this is a chance to have a go at parallelising the tasks.

When working with source control, it is a good idea to keep the main branch deployable, that is, you shouldn’t be working on it directly. This is what code branches are for. You make your changes in the branch then, once you are happy, you can merge the branch back in to the main

branch ready for deploying.

It is a good idea to not work on the same part of the code in different branches as this can lead to conflicts! Therefore, it is recommended that edits to a code file is only done by one person, using one branch, at any one time until the edits have been merged into the main.

So each of you can create a branch to tidy a specific file.

The git checkout command is used to switch between branches. If you use it with the -b argument, then you can also create a new branch before switching to it.

In this example, we will create a branch to tidy up the using statements in the sprite class. **Note, only one of you should do this with the Sprite class. Each of you should do this with a different class file.**

In the command prompt, type “git checkout -b tidy_usings_sprite”. You should use an appropriate variant branch name for the file that you will be editing.

You can then use the “git branch” command to verify that you are now in that new branch.

```
E:\Student GitHubs\700111\New folder\700111-2122-cracker-chase-crackertest>git checkout -b tidy_usings_sprite
Switched to a new branch 'tidy_usings_sprite'

E:\Student GitHubs\700111\New folder\700111-2122-cracker-chase-crackertest>git branch
  main
* tidy_usings_sprite
```

Now you can open the Visual Studio solution.

Navigate to the file that you want to edit. In the case of this example, it is the Sprite.cs file. Notice the block of using statements at the top of the file. Some of them should be greyed out because they are not being used. It may take a moment for the file to be fully parsed before this effect is visible.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

Delete all of the unused using statements. Then save the file.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using System;
```

Let’s return to the command prompt.

First we will use the git status command to see what changes have been made. Type “git status” in to the command prompt.

```
C:\CrackerChase>git status
On branch tidy_usings_sprite
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   CrackerChase/Sprite.cs

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .vs/
        CrackerChase/obj/

no changes added to commit (use "git add" and/or "git commit -a")
```

Note that it shows two sections. Changes that are not staged, and Untracked files.

Untracked files are files that are not versioned. That is that they are not included amongst the list of files that git is looking after.

The other section shows tracked files that have been modified, but are currently not flagged for having those changes committed.

We will use the git add command to stage the file we changed for committing to the repo.

Type “git add CrackerChase/Sprite.cs” in to the command prompt, changing it as necessary for the file that you changed.

There is no feedback on execution of this, so type “git status” again and not the difference.

```
C:\CrackerChase>git add CrackerChase/Sprite.cs

C:\CrackerChase>git status
On branch tidy_usings_sprite
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   CrackerChase/Sprite.cs

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .vs/
        CrackerChase/obj/
```

Note that you can also use the git add command to add untracked files. So if you create a new class file, for example, you can add it.

A note on untracked files. The .vs folder and the CrackerChase/obj folders are both things that should not really be in the repo. This is because they are autogenerated and would just clutter things. For the moment just ignore them. If you are feeling a bit more adventurous, then Google gitignore. This is a file that you can make that will include a list of the paths of all the files and folders that you want git to ignore. That way they will stop appearing in the lists.

Ok, so back to the modified file. Similarly to other source control systems like Subversion, a modified (or simply added) file is still not part of the repository until those changes have been committed. So that is what we are going to do next.

Type “git commit -m “ in to the command prompt and add between quotes a log message to accompany the commit. The log message is important for navigating changes in the repository. Do not leave it blank.

I used “Removed the unused using statements from the Sprite class”.

```
E:\Student GitHubs\700111\700111-2122-cracker-chase-crackertest>git commit -m "Removed the unused using statements from
the Sprite class"
[main a1e2734] Removed the unused using statements from the Sprite class
 1 file changed, 4 deletions(-)
```

Try using the “git log” command again. You should now see our new commit on our branch.

```
E:\Student GitHubs\700111\700111-2122-cracker-chase-crackertest>git log
commit a1e27349ca77e58f81bbc6f739c77bc0b2fc0ac9 (HEAD -> main)
Author: DavidParkerDr <david@goparker.com>
Date:   Fri Jan 28 16:50:19 2022 +0000

    Removed the unused using statements from the Sprite class

commit 4f61683ac397a60c7b8f84a69826e96ed9edf5f7 (origin/main, origin/HEAD)
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date:   Fri Jan 28 16:40:19 2022 +0000

    Initial commit
```

It is worth pointing out at this stage that the commit has been made to our local repository, and not on the remote one. In order to change that, we need to use the “git push” command. Type that in to the command prompt and press enter.

It will fail.

```
C:\CrackerChase>git push
fatal: The current branch tidy_usings_sprite has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin tidy_usings_sprite
```

This is because there isn’t a matching branch on the remote repository. Luckily it has suggested a solution.

Type “git push --set-upstream origin tidy_usings_sprite” making sure that the last bit matches your branch name.

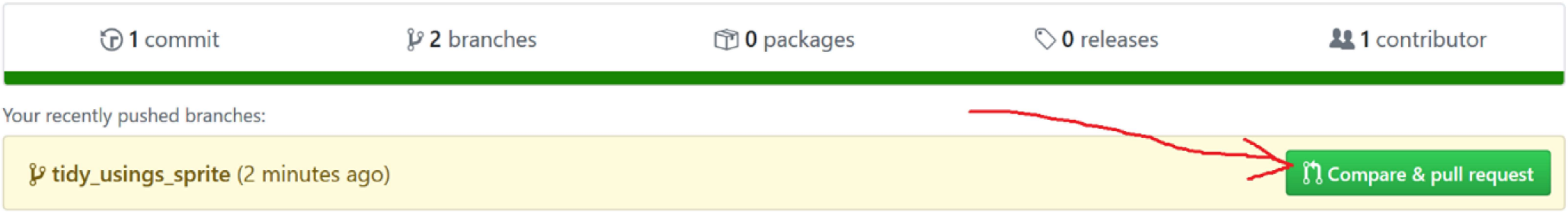
This will push the changes to the remote repository.

```
E:\Student GitHubs\700111\New folder\700111-2122-cracker-chase-crackertest>git push --set-upstream origin tidy_usings_sp
rite
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 378 bytes | 378.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
remote:
remote: Create a pull request for 'tidy_usings_sprite' on GitHub by visiting:
remote:      https://github.com/University-of-Hull-CST/700111-2122-cracker-chase-crackertest/pull/new/tidy_usings_sprite
remote:
To https://github.com/University-of-Hull-CST/700111-2122-cracker-chase-crackertest.git
 * [new branch]      tidy_usings_sprite -> tidy_usings_sprite
Branch 'tidy_usings_sprite' set up to track remote branch 'tidy_usings_sprite' from 'origin'.
```

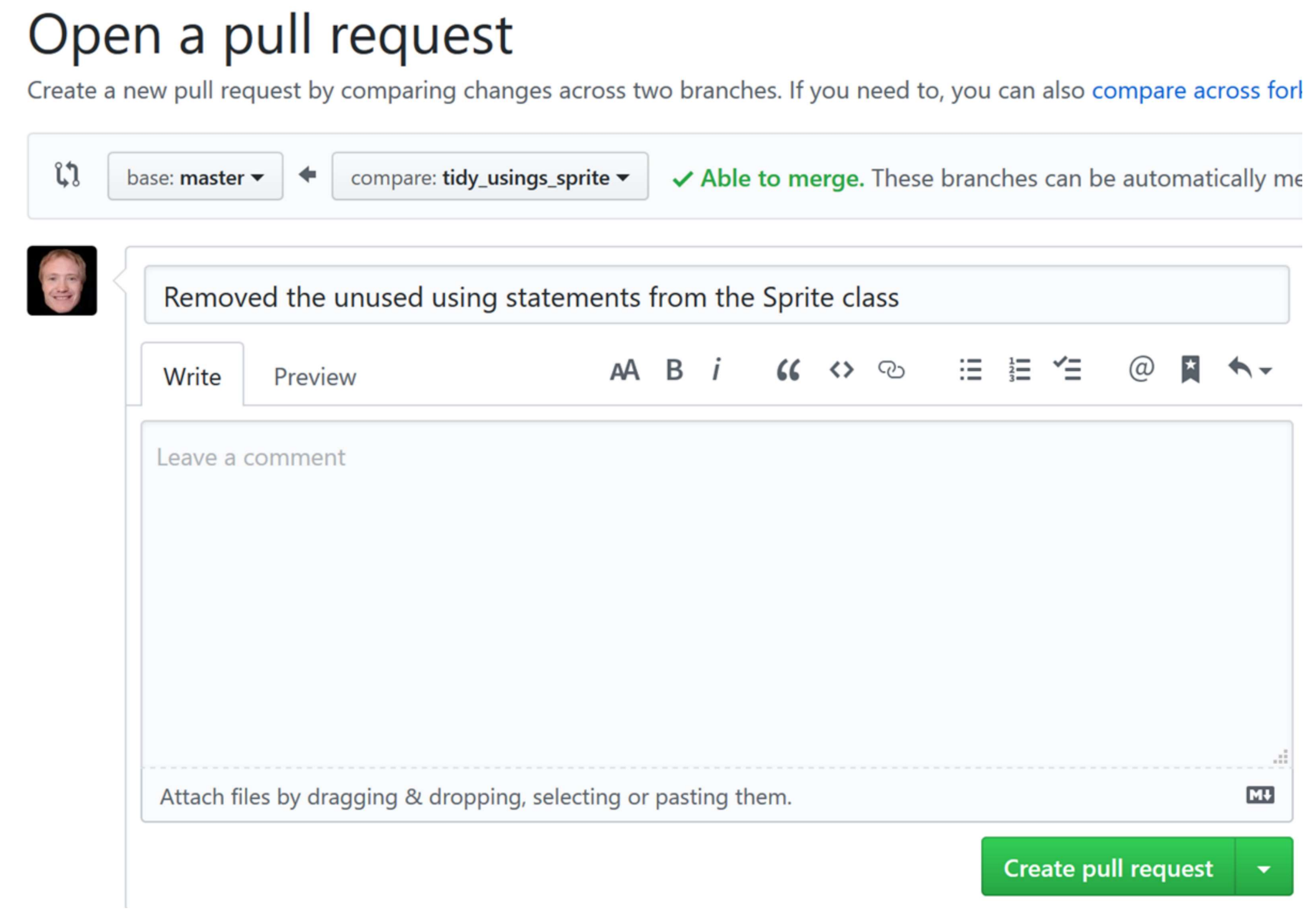
If you visit the repository page on GitHub, you should notice an update.

Next we want to create what is called a Pull Request. This is where we flag our changes as being ready to be merged in to the main branch.

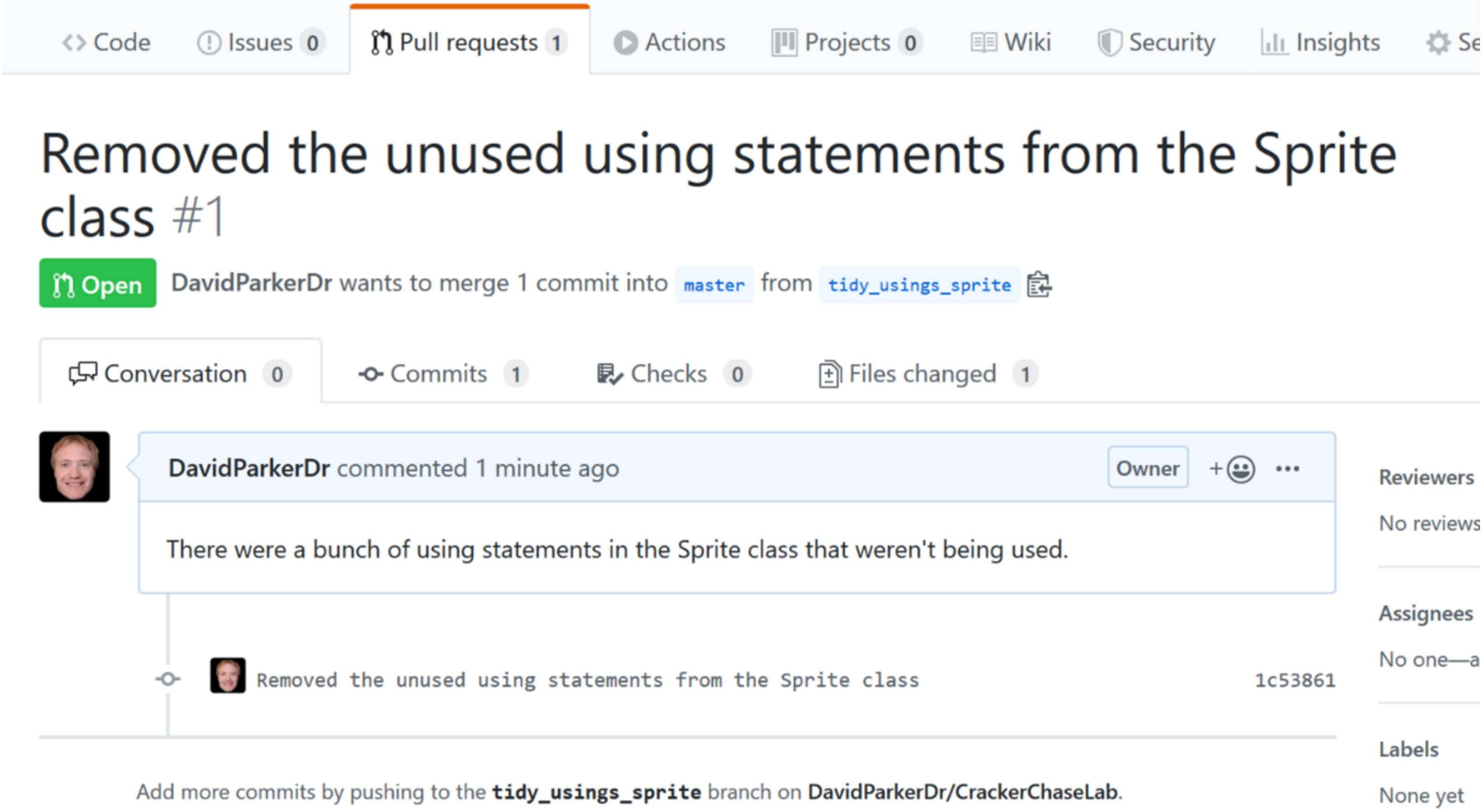
Click on the green button to the right of the branch. It is labelled Compare & pull request.



This will open a new page where you can write a description. Once you have done so, click on the Create pull request button.



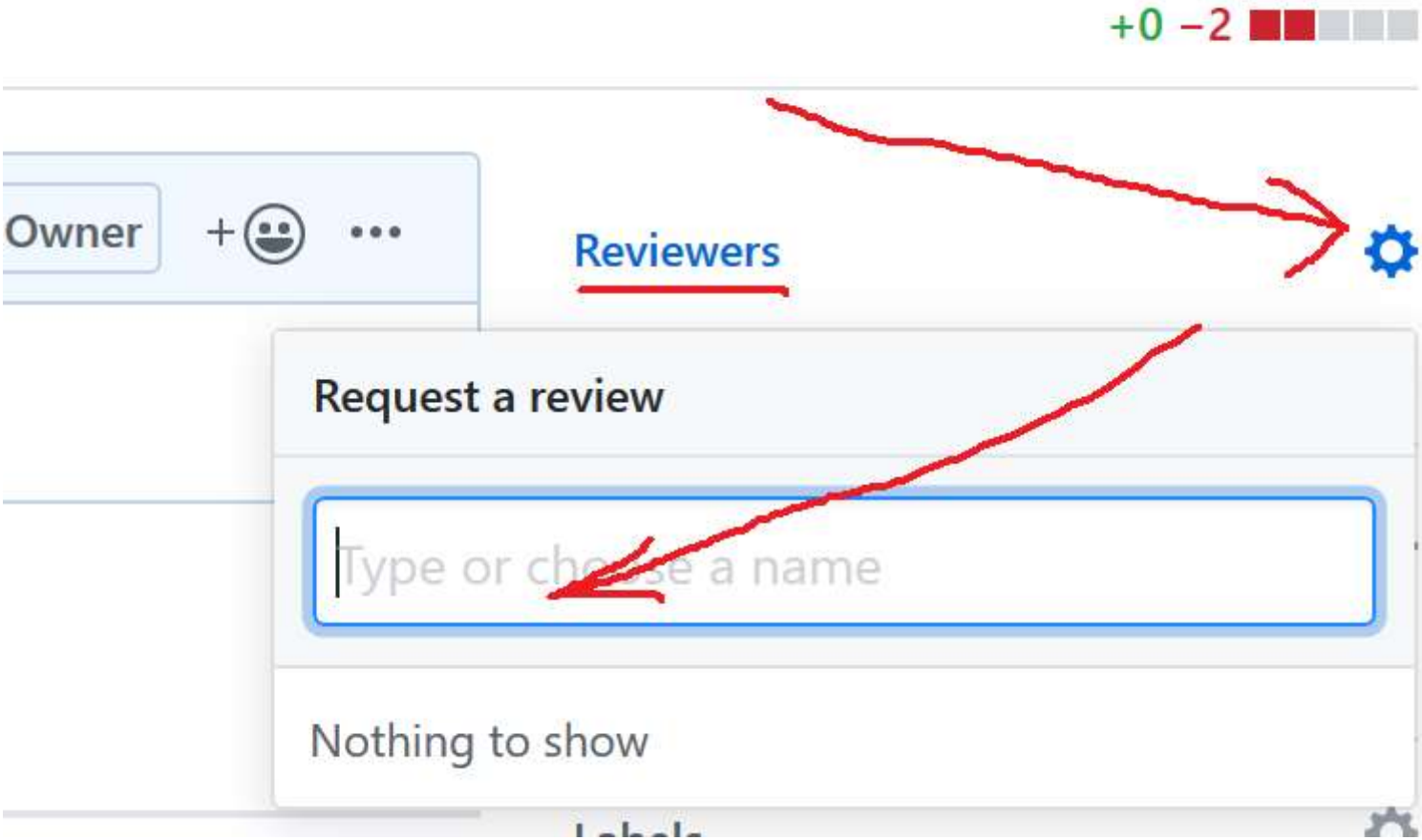
You should be taken to the list of pull requests page and be able to see the submitted request.



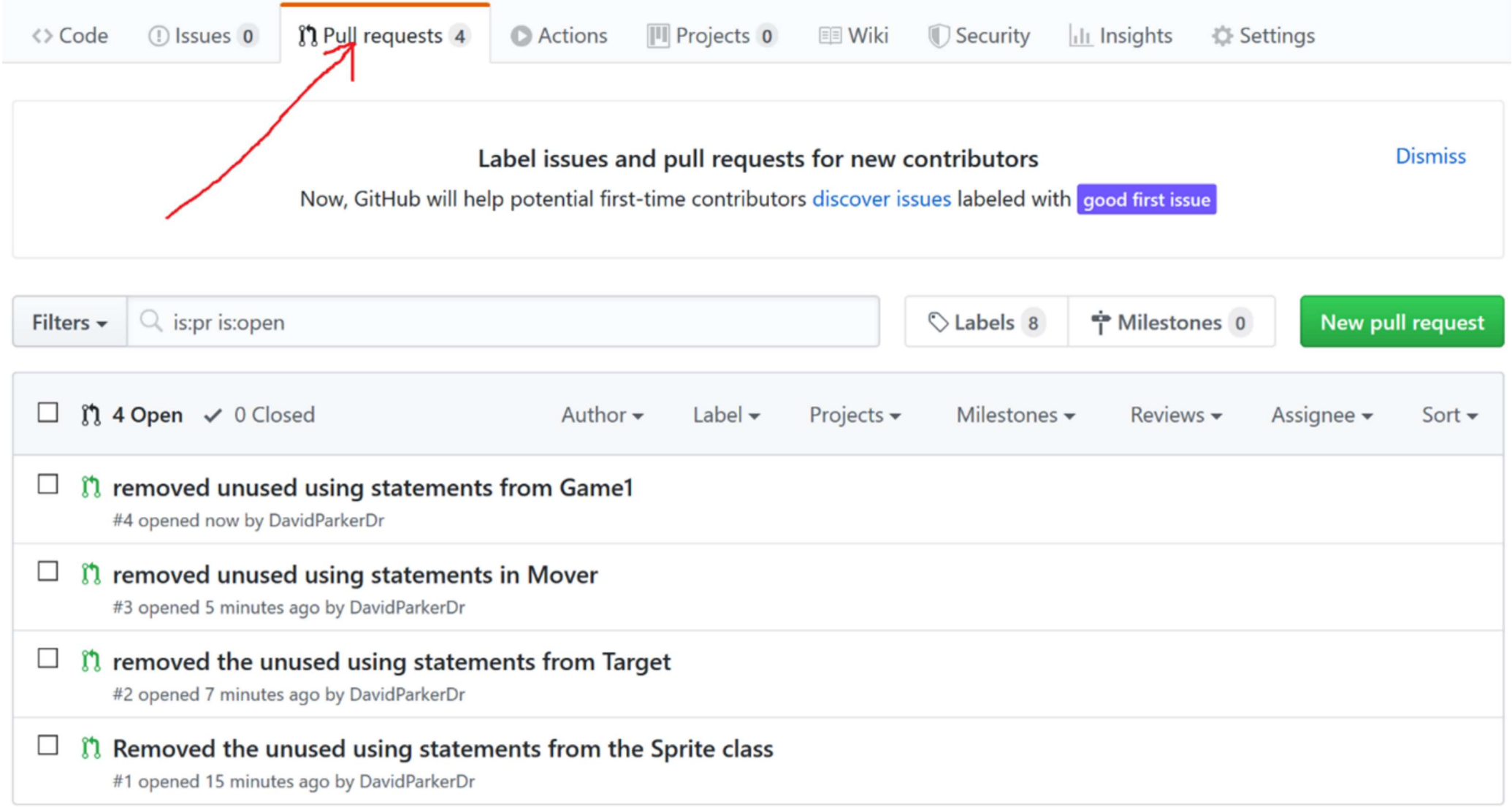
Make sure that everyone in the team has done this with their branches before moving on to the next step.

Though you are all collaborators, and could technically approve your own pull requests, it is generally better to have someone else review the changes. Consider designating one person on the team to handle all the requests.

When you have decided who this will be, you should add them as a reviewer on your pull request. Look to the right of the page and you will see a section “Reviewers” and a cog icon. Click on the cog icon and type in the reviewers username.

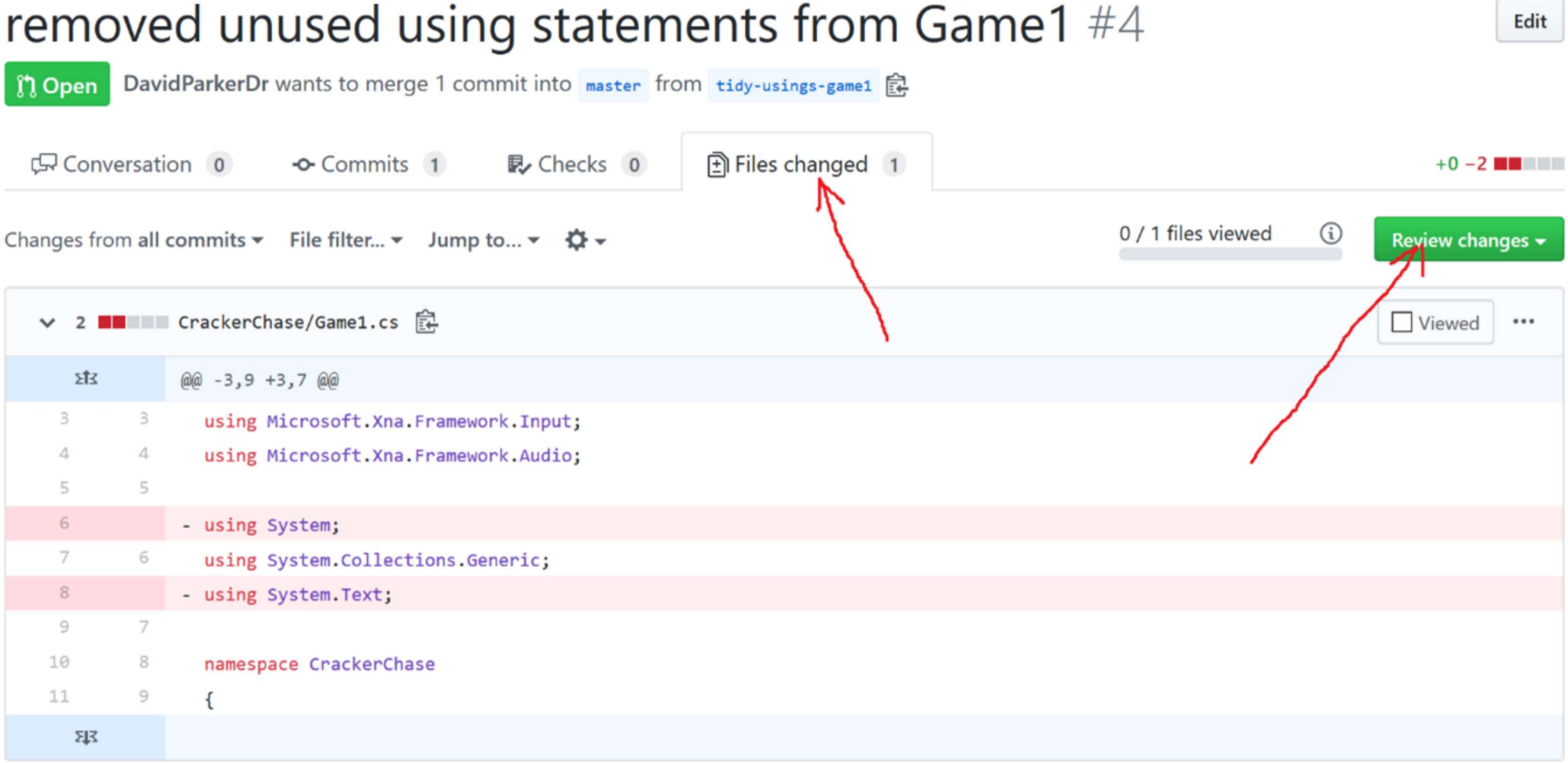


The reviewer should be alerted to review requests. If they click on the Pull requests tab they should also see a list of all the pull requests.

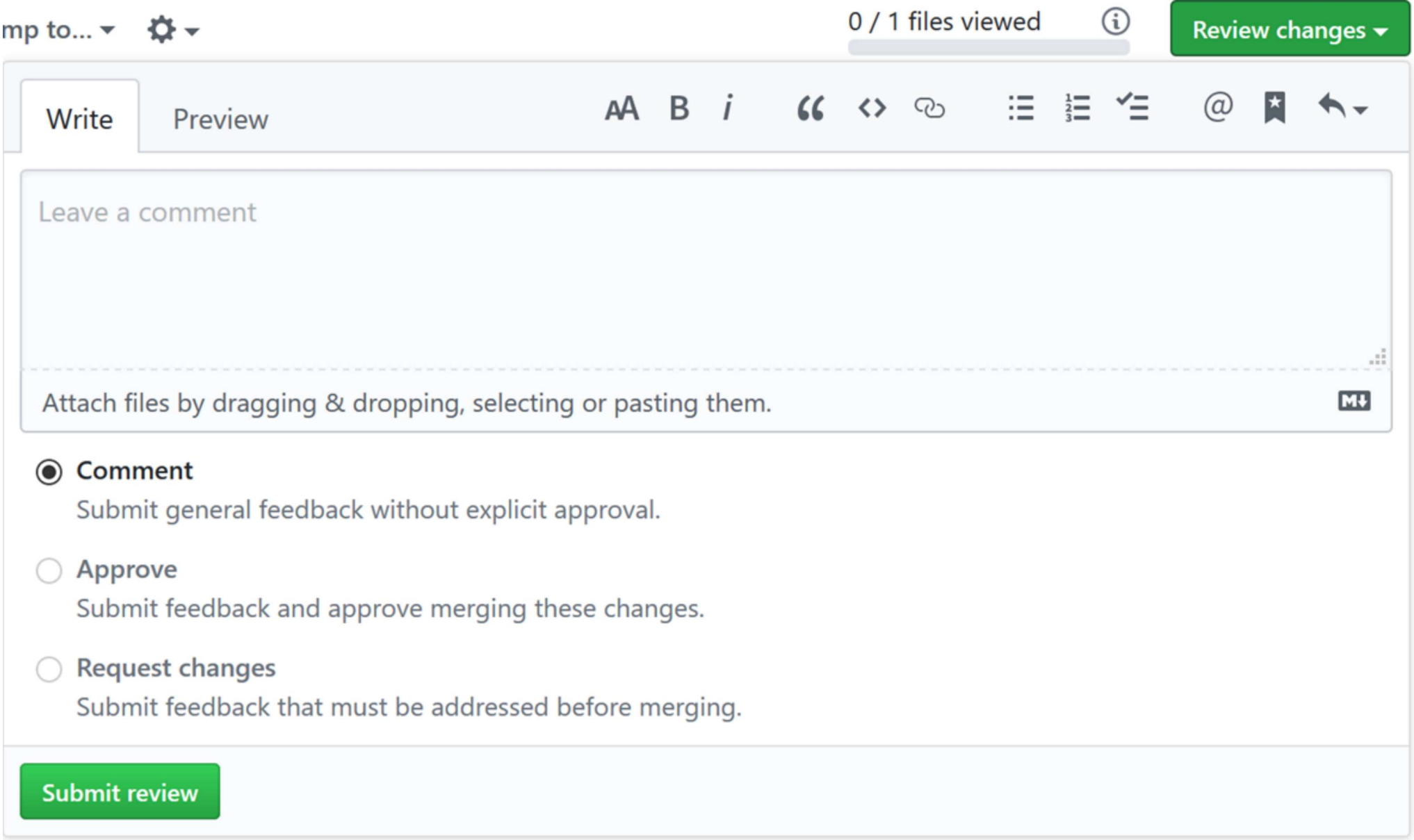


One at a time, click on a pull request.

If you click on the Files changed tab it should reveal the changes made to the files. The reviewer should also click the review changes button on the right.



This will pop-up a field to comment on the changes and either approve or not.



It is worth noting that you are not able to Approve your own Pull request.

Back on the Pull requests page, you can click on the Conversation tab.

If you are happy to merge the pull request then you can click on that button.

If you want to completely reject it, then you can add a comment and click the close request button lower down.

A good way of working would be to gather together as a group and go through the pull requests together. That way everybody is kept up to date on what is happening. You also get more eyes on the changes to ensure that they are good.

Conversation 0 Commits 1 Checks 0 Files changed 1

DavidParkerDr commented 30 minutes ago Owner

There were a bunch of using statements in the Sprite class that weren't being used.

`Removed the unused using statements from the Sprite class` `1c53861`

Add more commits by pushing to the **`tidy_usings_sprite`** branch on **DavidParkerDr/CrackerChaseLab**.

Continuous integration has not been set up

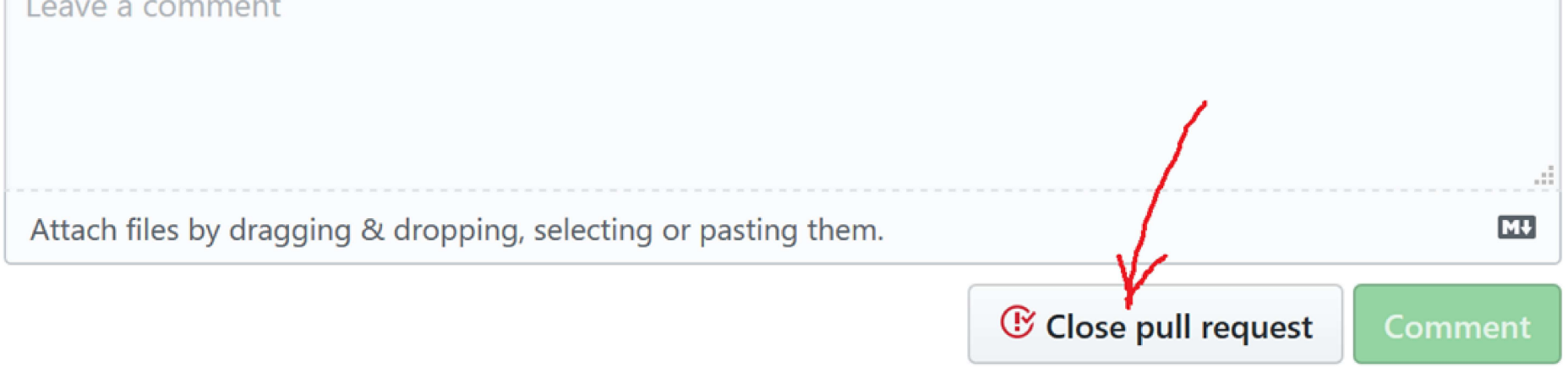
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request You can also open this in GitHub Desktop or view command line instructions.

Write Preview



You should get a confirmation of your merged pull request along with a button to delete the branch. You should delete the branch as it is no longer required.

Once the pull requests have been accepted you can checkout your local main branch again.

Type “git checkout main” into the command prompt.

Now we are going to pull the accepted changes from the remote repository down to our local copy.

Use the command “git pull” for this.

```
E:\Student GitHubs\700111\New folder\700111-2122-cracker-chase-crackertest>git pull
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 664 bytes | 73.00 KiB/s, done.
From https://github.com/University-of-Hull-CST/700111-2122-cracker-chase-crackertest
   4f61683..af161a4  main       -> origin/main
Updating 4f61683..af161a4
Fast-forward
 CrackerChase/Sprite.cs | 4 ----
 1 file changed, 4 deletions(-)
```

It will grab all of the changes and show you a summary.

Use the “git branch” command to show you your branches.

```
E:\Student GitHubs\700111\New folder\700111-2122-cracker-chase-crackertest>git branch
* main
  tidy_usings_sprite
```

Now that the pull request has been accepted, we can delete our local branch.

Use the command “git branch -d tidy_usings_sprite”.

```
E:\Student GitHubs\700111\New folder\700111-2122-cracker-chase-crackertest>git branch -d tidy_usings_sprite
Deleted branch tidy_usings_sprite (was 33be3d5).
```

You should now be left with just the main branch which should contain all of the changes from the different pull requests.

And now for the rest

The remainder of this document has a bunch of different tasks for improving the CrackerChase game. You may also be able to think of some of your own. Read though the document as a group and try and identify where the work can be split in to branches to be worked on separately. Remember about avoiding the crossing of the streams.

Once you have done that, assign the tasks and get on with it. Help each other out and keep on top of the changes.

Run the sample

Open up the Visual Studio solution in the Crackerchase folder. Have a little look around.

Try running it. It should compile and run.

You can use the arrow keys to move the cheese around the screen. The idea is that you collect as many crackers as possible before the time runs out. Once the time runs out it will display your score and when you press enter, your game will end.

Add some states

As it stands, it is pretty annoying that after every game you have to exit the program and start again.

Define an enum in your Game1 class giving it a meaningful name such as GameStates. You will want to specify 2 game states inside it, one for displaying a start screen, and the other for when the game is playing.

Once the enum is defined, you can declare one (also in the Game1 class) that will store the current state. You should give it a meaningful name such as ‘state’ or ‘currentState’. For now, it will be uninitialized.

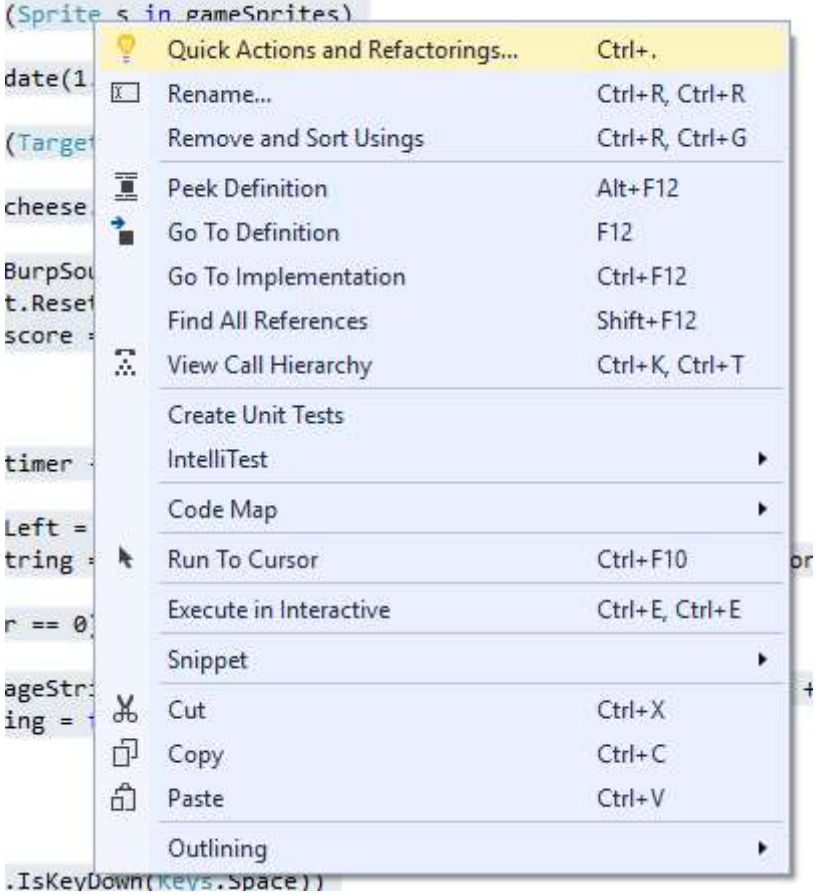
You will want to initialise it in the LoadContent method. At the moment you only have code for the gameplay state so set it to that for now. Obviously later, when the game first opens, you will want the state to start on the start screen state.

Refactor your update and draw methods

Previously your update and draw methods only had one job each to do. Now, with the addition of different states to your game, they are going to have to do some extra work. It would be bad practice for us to try and cram all the required functionality in to those methods as it would be cluttered and difficult to navigate and maintain.

What we are going to do is refactor those methods in to sub methods that we can then call from the original method. The code in there at the moment is concerned only with the gameplay, so I suggest that you create new methods called ‘drawGamePlay’ and ‘updateGamePlay’.

You can do this easily using the refactor tools in Visual Studio. First select all of the code in Update (except the call to the base.Update method). Right click on it. The top option is Quick Actions and Refactorings. You should click on this.



Then you should click on the Extract Method option.

This will extract all of your selected code into a new method. Note that the NewMethod name will be highlighted in green. As you start typing, both references will update. Give your new method an appropriate name as suggested above.

```
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</par
0 references
protected override void Update(GameTime gameTime)
{
    NewMethod();

    base.Update(gameTime);
}

1 reference
private void NewMethod()
{
    KeyboardState keys = Keyboard.GetState();

    if (playing)
    {
```

Run your game again. It should still work exactly as before.

Now back in your Update and Draw methods add a conditional (an if or switch statement) that queries the value of the current state variable that you made before. If it is set to the game play state, then call the gameplay versions of those methods.

You will also want to make sure that there is an option for the other state, but for now there is nothing to call because you don't have any alternative code, but you are going to fix that next.

Add new methods for updating and drawing the start state

Now that we are going to be using the states you can remove the 'playing' variable from the Game1 class. So go through the code and remove all references to it. There will be a definition near the top, an initialising in the startPlayingGame method, and the if statement in your new updateGameplay method. You will want to keep the code inside the if statement, but you can remove the code in the else clause. This should leave you with the bulk of the code in the updateGameplay method, but it won't be inside and if statement now.

Make a new method and call it something like updateStartScreen. This method has a very simple task, to monitor for the space bar being pressed in order to trigger a new game.

Have a look inside the updateGamePlay method for a clue on how to query the key states. You can borrow code from there in your new method. The key state you are going to be using is 'Keys.Space'. When you detect that the space bar is down, call the startPlayingGame method.

This would be a good time to change the initial state of the game in the LoadContent method to the start screen state (whatever you called it). You can also remove the call to startPlayingGame in the LoadContent method.

Instead you can initialise the messageString variable to something like "Press space to start the game".

For your new draw method, I would start for now by copying and pasting your drawGamePlay method and renaming it to drawStartScreen. The only real difference between the two is that you want to change the statusPos variable so that it draws the messageString in the centre of the screen.

```
Vector2 statusPos = new Vector2(xPos, screenHeight / 2);
```

It should already be centred left to right and the above code replacement will put it in the middle top to bottom.

Remember to fix your Draw and Update method so that it will call the right sub method depending on what the state is.

Run it and see what happens.

It should run and start by displaying a message in the middle of the screen telling you to press the space bar.

When you press the space bar, the game should start.

The game should play as normal, but now when the time reaches zero, there is nothing to tell it to stop so it will go in to a negative time and keep going.

Game over man, game over

Let’s add a method called gameOver. We are going to call this method when the time runs out in our updateGamePlay method.

The game over method is going to set the current state to the startScreen state. It is also going to change the messageString so that it tells you that the game is over and that you will need to press the space bar to start again. You should also get it to tell you what the score is.

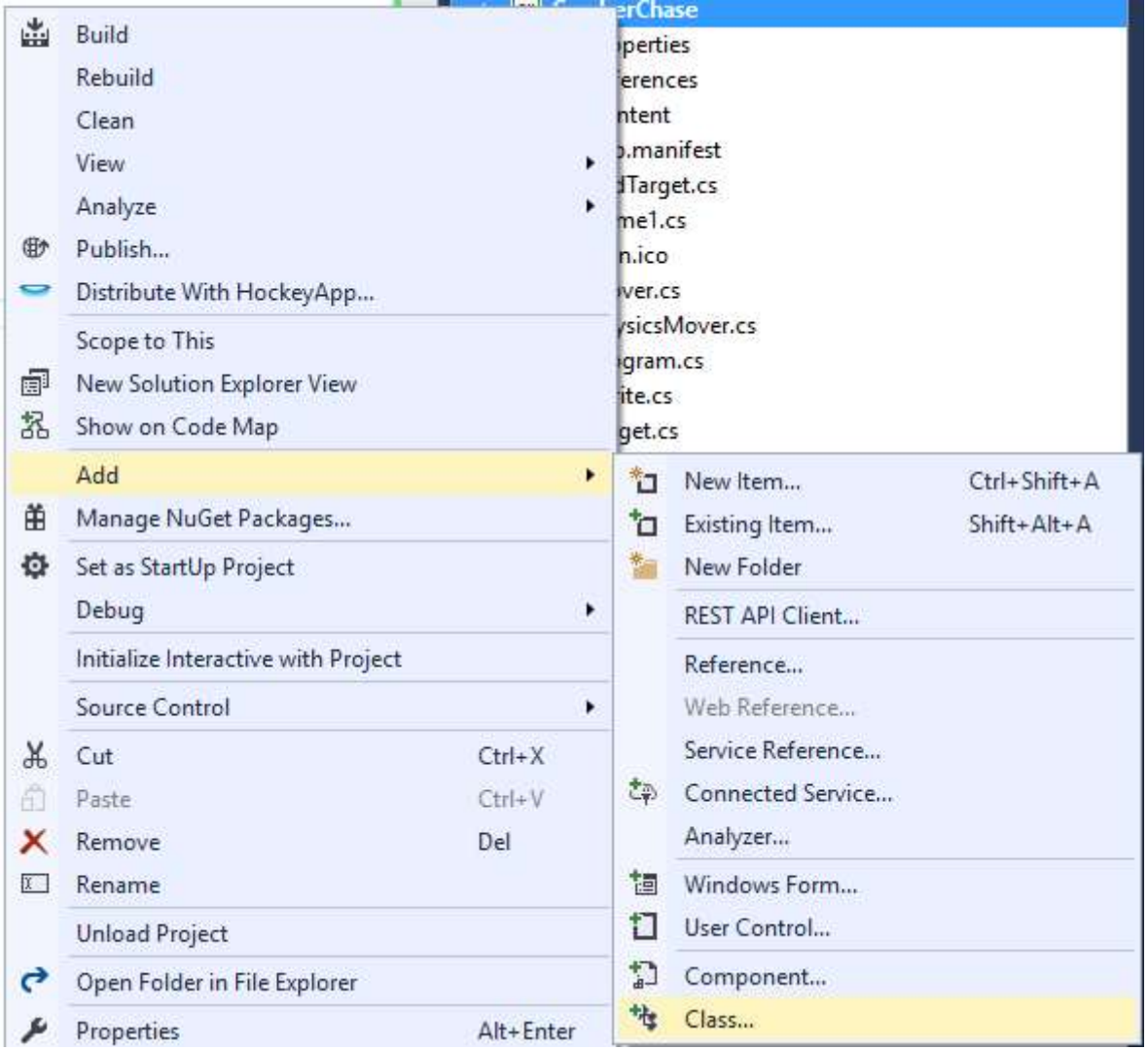
Run it and see what happens.

Now you should be able to start the game and get invited to press the space bar to begin the game. When you press the space bar, the state changes and you can move the cheese around to gain score. When the time runs out the game finishes and displays a message showing you your score and inviting you to start again. You should be able to play as many games as you can stand.

Let’s make the game a bit more interesting

At the moment there is only one type of target. It is pretty boring. Where is the jeopardy?

Right click on the CrackerChase project line in your Solution Explorer (it is the one that holds all of the class files. Then navigate to Add > Class and click it.



It will prompt you for a name. Call it something like BadTarget.

This will create a new class file of that name. You will want to update the class definition so that it inherits from the existing Target class.

You do this by adding a colon after the class name, followed by the name of the class you want to inherit from.

```
class BadTarget : Target
```

You will also need a constructor for the class. For now, copy and paste the one from the Target class and rename it to BadTarget.

```
public BadTarget(int inScreenWidth, int inScreenHeight, Texture2D inSpriteTexture, int inDrawWidth, float inResetX, float inResetY)
   : base(inScreenWidth, inScreenHeight, inSpriteTexture, inDrawWidth, inResetX, inResetY)
{

```

```
    // stuff goes in here
}
```

You will need to add some ‘using’ definitions at the top of the class file so that it knows what a Texture2D is.

```
using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;
```

Back in the Target class we want to add a new property. It will be called Score and you can add it above the definition of the Random variable.

```
public int Score { get; set; }
```

This is going to allow us to specify a score for collecting a particular target.

In the constructor for the Target class add a statement that sets the Score to 10.

In the constructor for the BadTarget class, add a statement that sets the Score to -100.

Note that we don’t need a definition for Score in the BadTarget class because it can inherit it from its parent Target class.

Back in the Game1 class find the bit of code in the update methods that deals with what happens when a collision occurs. Change the statement adding 10 to the score to instead add t.Score to it. This will mean that whichever object type is collided with will update the score accordingly.

We are probably going to want to have some way of identifying the BadTargets on screen so let’s make them red.

In the BadTarget class, we are going to override the Draw method to make it behave differently to its parent. Copy and paste the Draw method from the Target class in to the BadTarget class. Change the word ‘virtual’ to ‘override’. This will tell the object to use its own method instead of the parent one.

Change the colour in the spritebatch.Draw call from Color.White to Color.Red.

Run it and see what happens.

It should run. But you won’t see any differences as we haven’t actually made any BadTargets yet. We will fix that next, but first…

Constructing some BadTargets

In the LoadContent method of the Game1 class there is a for loop that constructs new Targets and adds them to a list of crackers and gamesprites.

Copy and paste that loop below itself. Change the number of iterations of the new loop to 10 instead of 100 and change the type of the constructed object to BadTarget.

The compiler should not complain about this because a BadTarget is still a type of Target (it inherits from it) and consequently is also a type of Sprite.

Run it and see what happens.

Now when you run it, there should be some red crackers in amongst the yellow ones. When you collide with them you should get a reduction in your score.

Challenge

Play around a bit more. See if you can make some other interesting gameplay changes. Make sure that you commit any changes that you make.