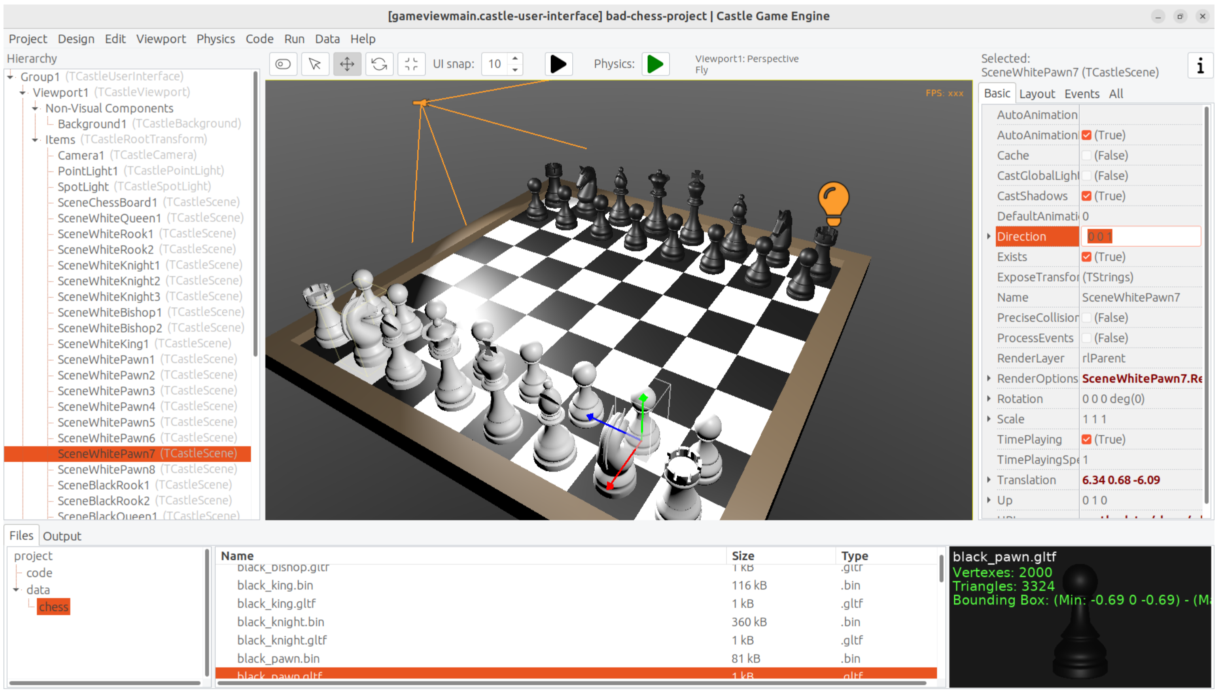
The bad way to play chess

3D physics fun using Castle Game Engine (Part 1)

Michalis Kamburelis

Table of Contents

1. Introduction 1
2. The Real Introduction 2
3. Download and install the engine 3
4. Create your first project 4
5. Optionally tweak the editor preferences 8
6. Learning to design 3D items in a viewport 9
7. Design a 3D chessboard with chess pieces 12
8. Using physics in the editor 16
9. Summary 22



[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project
Design
Edit
Viewport
Physics
Code
Run
Data
Help
Hierarchy
UI snap: 10
Physics:
Viewport1: Perspective
Fly
FPS: xxx
Group1 (TCastleUserInterface)
Viewport1 (TCastleViewport)
Non-Visual Components
Background1 (TCastleBackground)
Items (TCastleRootTransform)
Camera1 (TCastleCamera)
PointLight1 (TCastlePointLight)
SpotLight (TCastleSpotLight)
SceneChessBoard1 (TCastleScene)
SceneWhiteQueen1 (TCastleScene)
SceneWhiteRook1 (TCastleScene)
SceneWhiteRook2 (TCastleScene)
SceneWhiteKnight1 (TCastleScene)
SceneWhiteKnight2 (TCastleScene)
SceneWhiteKnight3 (TCastleScene)
SceneWhiteBishop1 (TCastleScene)
SceneWhiteBishop2 (TCastleScene)
SceneWhiteKing1 (TCastleScene)
SceneWhitePawn1 (TCastleScene)
SceneWhitePawn2 (TCastleScene)
SceneWhitePawn3 (TCastleScene)
SceneWhitePawn4 (TCastleScene)
SceneWhitePawn5 (TCastleScene)
SceneWhitePawn6 (TCastleScene)
SceneWhitePawn7 (TCastleScene)
SceneWhitePawn8 (TCastleScene)
SceneBlackRook1 (TCastleScene)
SceneBlackRook2 (TCastleScene)
SceneBlackQueen1 (TCastleScene)
Basic
Layout
Events
All
Files
Output
project
code
data
chess

Selected:
SceneWhitePawn7 (TCastleScene)

AutoAnimation	
AutoAnimation	✓ (True)
Cache	(False)
CastGlobalLigh	(False)
CastShadows	✓ (True)
DefaultAnimati	0
Direction	0 0 1
Exists	✓ (True)
ExposeTransfor	(TStrings)
Name	SceneWhitePawn7
PreciseCollisior	(False)
ProcessEvents	(False)
RenderLayer	rlParent
RenderOptions	SceneWhitePawn7.Re
Rotation	0 0 0 deg(0)
Scale	1 1 1
TimePlaying	✓ (True)
TimePlayingSpe	1
Translation	6.34 0.68 -6.09
Up	0 1 0

Name	Size	Type
black_bishop.gltf	1 kB	.gltf
black_king.bin	116 kB	.bin
black_king.gltf	1 kB	.gltf
black_knight.bin	360 kB	.bin
black_knight.gltf	1 kB	.gltf
black_pawn.bin	81 kB	.bin
black_pawn.gltf	1 kB	.gltf

black_pawn.gltf
Vertexes: 2000
Triangles: 3324
Bounding Box: (Min: -0.69 0 -0.69) - (M

1. Introduction

I remember my first book about chess, when I was a kid. It was a book teaching young people how to play chess. The first chapter started with a tale about children *playing* *chess incorrectly*: they didn’t know the rules, so they put chess pieces randomly on the chessboard, and flicked them with their fingers towards the other side. The wooden chess pieces flew in the air, bashed with each other. Eventually most of the chess pieces fell off the chessboard onto the floor. The person with the last chess piece remaining on the chessboard was the winner.

That was naturally a bad way to play chess. In the second chapter of the book, an adult came, told children that they play chess wrong, and taught them the *right way* — how each figure moves, how the king is special, what it means to check and then mate your opponent. The book overall was great, and it’s likely responsible for my love for chess (the *proper* version of the game, with rules instead of flicking objects) to this day.

That being said… Don’t you want to play some day this *"incorrect"* version of chess, the children’s version, where nothing else matters except just sending each chess piece flying toward the other side?

In this series of articles we will go back in time, erase our hard-earned knowledge about how to *really* play chess, and implement a simple 3D physics fun application where you can flick chess pieces using physics. You can treat it as a game for 2 people — just play it on a computer, and let each player use the mouse and keyboard in turn.

2. The Real Introduction

The real purpose of this article is to be an entertaining but also useful introduction to using *Castle Game Engine*.

Castle Game Engine is a cross-platform (desktop, mobile, consoles) 3D and 2D game engine. We will learn how to make a game for desktops (Linux, Windows, macOS, FreeBSD).

In the first part of the article we will show how to design a 3D chessboard and chess pieces using *Castle Game Engine* editor and how to use *physics*. In the next part, we will do some coding in Pascal to implement the game logic. In future articles we’d like to show also development for other platforms (like Android and iOS) and future plans (like the *web platform*).

You can use FPC or Delphi to develop the application presented here. In our engine, we are committed to perfect support for both of these Pascal compilers. Though note that with Delphi, you can right now target only Windows and Linux (with FPC, all the platforms are available).

Castle Game Engine features a powerful visual editor to design your games, in 2D or 3D. Just like Delphi and Lazarus visual libraries, it's all based on a simple RAD concept: you can design a functional application easily visually but at the same time **everything you do is actually using Pascal classes and properties**. So all your knowledge gained from using the editor is also useful when you need to write some Pascal code. You will use the same classes and properties in Pascal that you've seen in the visual editor.

The engine is free and open-source. Use it to develop open-source or proprietary applications. You can distribute them to friends in any way, you can publish them on Steam, Itch.io, Google Play (Android), AppStore (iOS), your own website — everywhere.

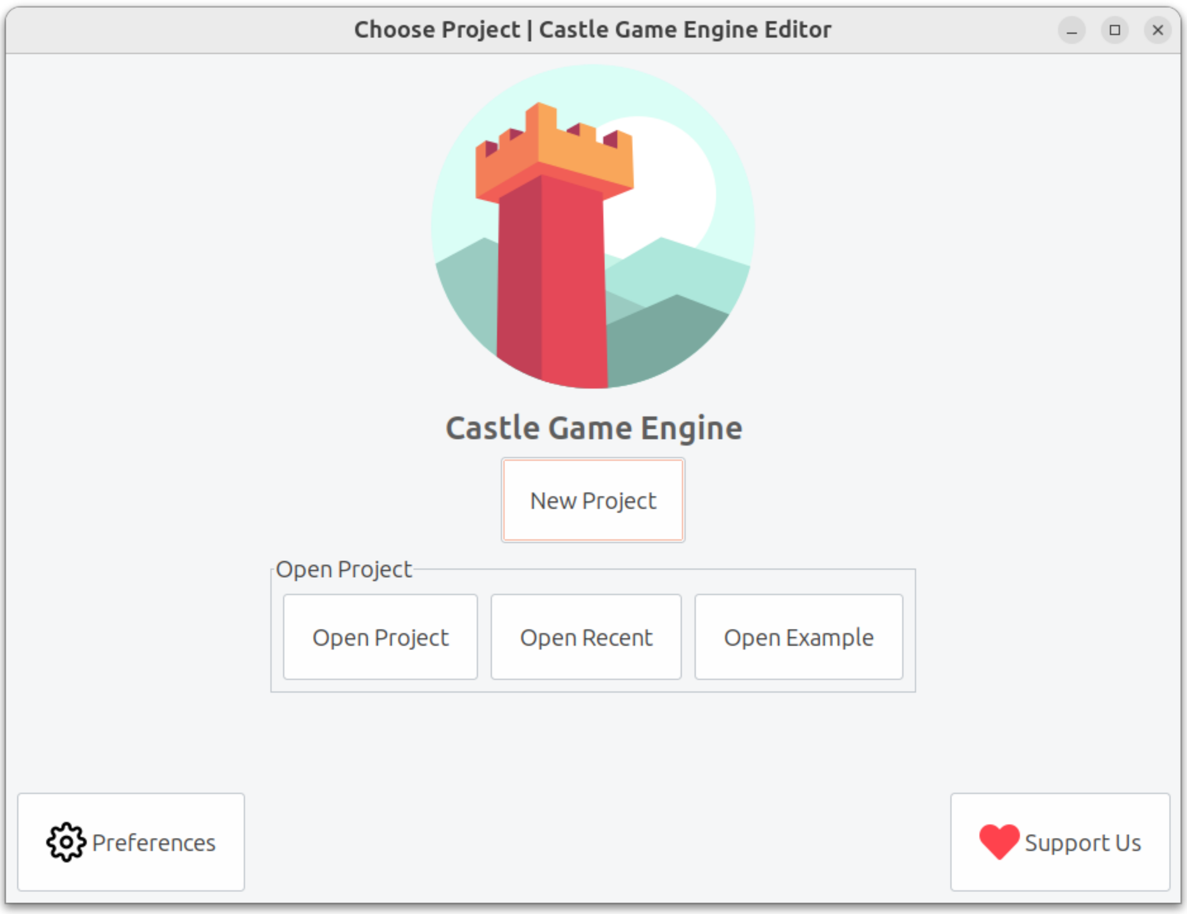
3. Download and install the engine

Start by downloading the engine from our website: `https://castle-engine.io/download` . Choose the version suitable for your operating system.

- On Windows, the recommended download is a simple installer. Just run it.
- On Linux, just unpack the downloaded zip file to any directory you like.
- Follow our website for more detailed instructions and other platforms.

Once installed, run the *Castle Game Engine* editor.

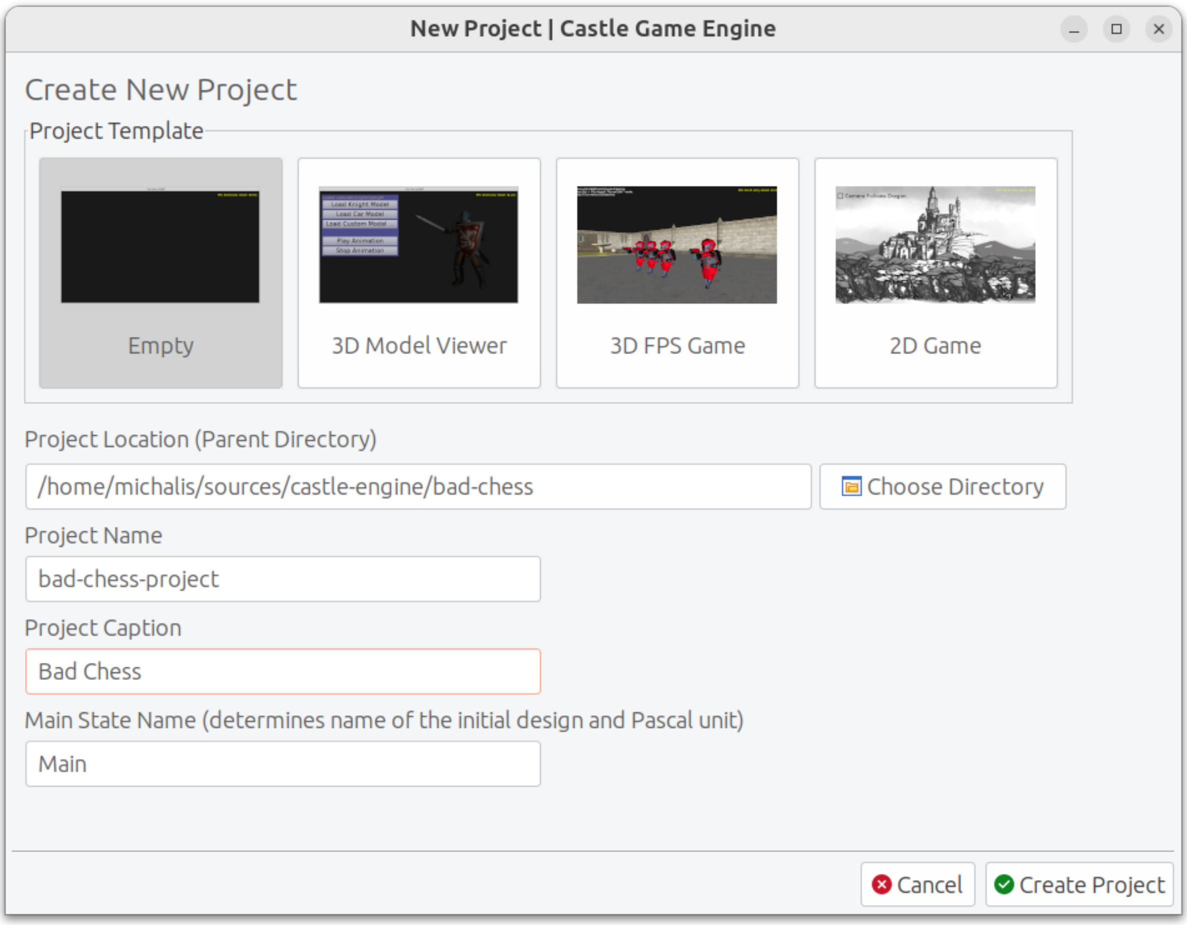
- If you used the installer on Windows, then the shortcut to run *Castle Game Engine* has already been created for you.
- If you unpacked the engine from a zip file, then run the binary `castle-editor` from the subdirectory `bin` where you have unpacked the engine.



If you encounter any issue, consult our manual on `https://castle-engine.io/install` .

4. Create your first project

Let’s create a new project. Click on the *"New Project"* button, choose the *"Empty"* project template, configure the project name and directory as you wish, and click *"Create Project"*.



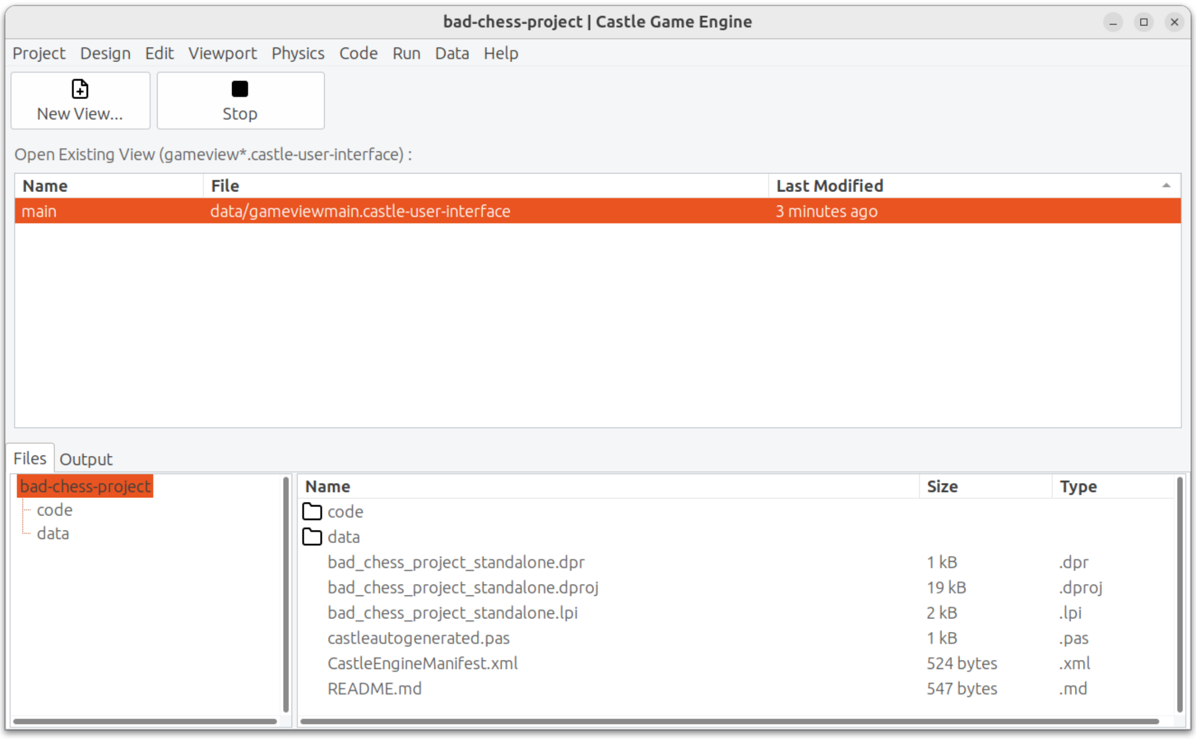
Project Location (Parent Directory)
/home/michalis/sources/castle-engine/bad-chess

Project Name
bad-chess-project

Project Caption
Bad Chess

Main State Name (determines name of the initial design and Pascal unit)
Main

In response, we will create a new directory with a few project files that define your project data and initial Pascal code.


bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
New View...
Stop
Open Existing View (gameview*.castle-user-interface) :
Files
Output
bad-chess-project
code
data

Name	File	Last Modified
main	data/gameviewmain.castle-user-interface	3 minutes ago

Name	Size	Type
code		
data		
bad_chess_project_standalone.dpr	1 kB	.dpr
bad_chess_project_standalone.dproj	19 kB	.dproj
bad_chess_project_standalone.lpi	2 kB	.lpi
castleautogenerated.pas	1 kB	.pas
CastleEngineManifest.xml	524 bytes	.xml
README.md	547 bytes	.md

You can explore the files in your project using the bottom panel of the editor. You can also just explore them using your regular file manager — there's nothing special about this directory, these are normal files and directories.

The most important files and directories are:

- `code` is a subdirectory where we advise to put all Pascal source code (units) of your application. Initially it contains just 2 units, `GameInitialize` and `GameViewMain`.
- `data` is a subdirectory where you should put all the data that has to be loaded at run-time by your application. All the 3D and 2D models, textures, designs have to be placed here if you want to use them in your game. Initially it contains the *design* called `gameviewmain.castle-user-interface` (and, less important, `CastleSettings.xml` and `README.txt` files).

The general idea is that the initial application (created from the *"Empty"* template) contains just a single *view* called `Main`. A *view* is a *Castle Game Engine* concept that represents something that can be displayed in a *Castle Game Engine* application. You use it typically quite like a *form* in Delphi or Lazarus. It is a basic way to organize your application.

- Every *view* can be visually designed. Just double-click on it, in the *"Open Existing View"* panel or in the *"Files"* panel (when you're exploring the `data` subdirectory).

This allows to visually design the contents of the `gameviewmain.castle-user-interface` file. The file has an extension `.castle-user-interface` because a *view* is a special case of *user interface* in Castle Game Engine.

In larger applications, you can have multiple views. Also, in larger applications, you can visually design some *user interface* elements that are not views, but are just reusable pieces of a user interface. All these files have the extension `.castle-user-interface` and can be visually designed using the editor. The views have, by convention, a name like `gameview*.castle-user-interface`.

- Every view has also an accompanying Pascal unit. The unit is named like the view, but without the `.castle-user-interface` extension. So in our case, the unit is called `gameviewmain.pas`. The unit contains the Pascal code that should be executed when the view is displayed. It defines a class that has virtual methods to react to various useful events (like view being started, or user pressing a key or a mouse button). You will often add more methods to it, to implement your application logic.

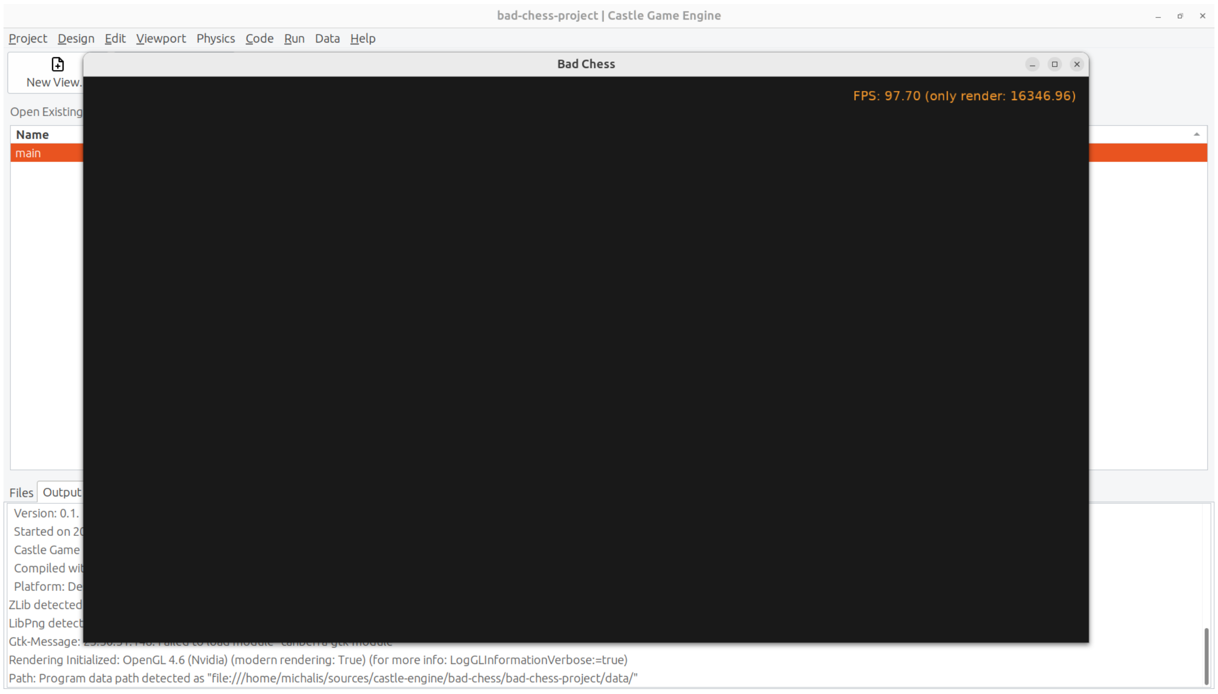
See `https://castle-engine.io/view_events` and `https://castle-engine.io/views` to learn more about the *views* in our engine.

To be clear about the terminology used throughout our engine:

- A *design* is a name for *a file you can visually design using our editor*. A design can be a file with extension:
  1. `.castle-user-interface` (user interface, can be loaded to a class descending from `TCastleUserInterface`)
  2. `.castle-transform` (3D or 2D transformation, can be loaded to a class descending from `TCastleTransform`)
  3. `.castle-component` (any other component; can be loaded to a class descending from `TComponent`)
- A *user interface design* is a specific case of a *design* file. It is a file with `.castle-user-interface` extension.
- A *view* is a specific case of a *user interface design*. By convention it is called like `gameview*.castle-user-interface`.

You're probably itching to start actually *doing* something after this lengthy introduction. Let's get to it.

As a first thing, make sure that everything works. Use the big *"Compile And Run"* button (key shortcut *F9*) and watch as the project is compiled and run. The result will be boring — dark window with FPS (*frames per second*) counter in the top-right corner. FPS are a standard way to measure your application performance.


bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
New View...
Open Existing
Name
main
Bad Chess
Files Output

```
FPS: 97.70 (only render: 16346.96)
```

```
Version: 0.1.
Started on 20
Castle Game
Compiled wit
Platform: De
ZLib detected
LibPng detect
Gtk-Message:
Rendering Initialized: OpenGL 4.6 (Nvidia) (modern rendering: True) (for more info: LogGLInformationVerbose:=true)
Path: Program data path detected as "file:///home/michalis/sources/castle-engine/bad-chess/bad-chess-project/data/"
```

5. Optionally tweak the editor preferences

Once things work, you may want to tweak them by going to editor *"Preferences"*. In particular:

- The editor by default uses a bundled version of latest stable *FPC (Free Pascal Compiler)*. If you'd rather use your own *FPC* installation or *Delphi*, configure it in the preferences.
- To edit the Pascal files, the editor by default tries to auto-detect various Pascal-capable IDEs and editors, like *Lazarus*, *Delphi*, *Visual Studio Code*. If you prefer to configure a specific editor, choose it in the preferences.

More details about the editor configuration can be found in our manual on `https://castle-engine.io/install` .

The editor can use any Pascal compiler and any text editor. We deliberately don't put any special requirements on what you can use. Though we make sure to support the popular choices perfectly. In particular, we have a dedicated support for using *Visual*

Studio Code with Pascal (and *Castle Game Engine* in particular), see `https://castle-engine.io/vscode` .

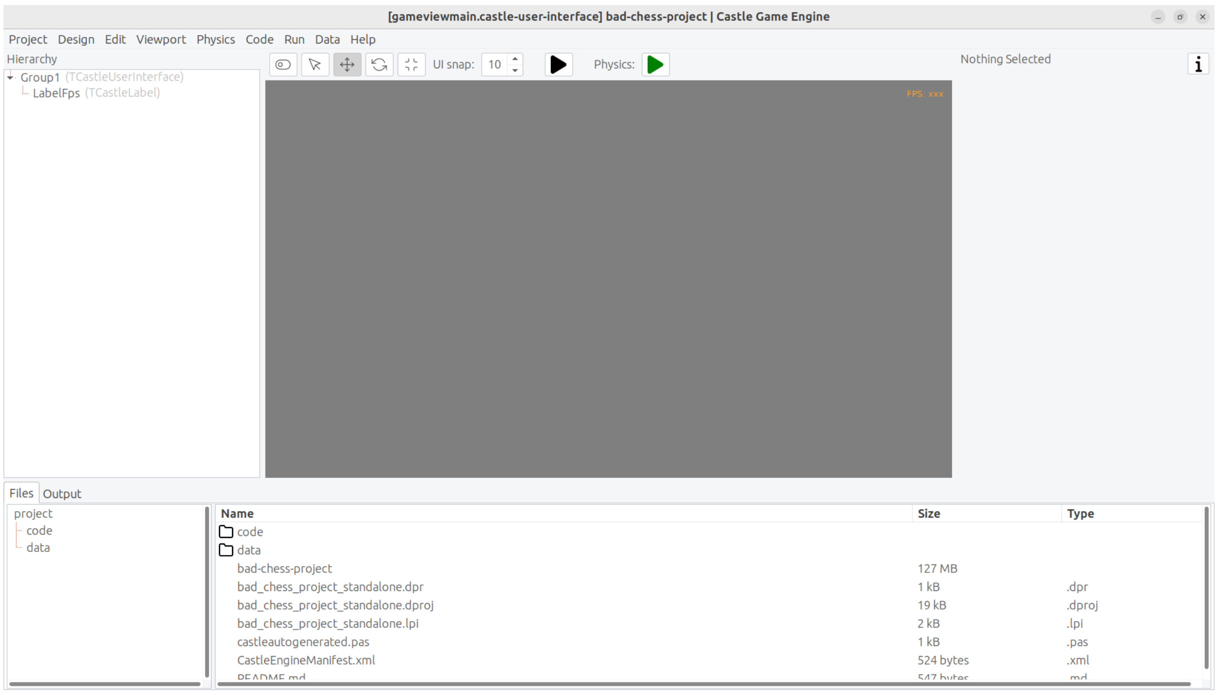
6. Learning to design 3D items in a viewport

If you haven't already, open the `main` view in the editor.

You can double-click on it in the *"Open Existing View"* panel or in the *"Files"* panel (when you're exploring the `data` subdirectory).

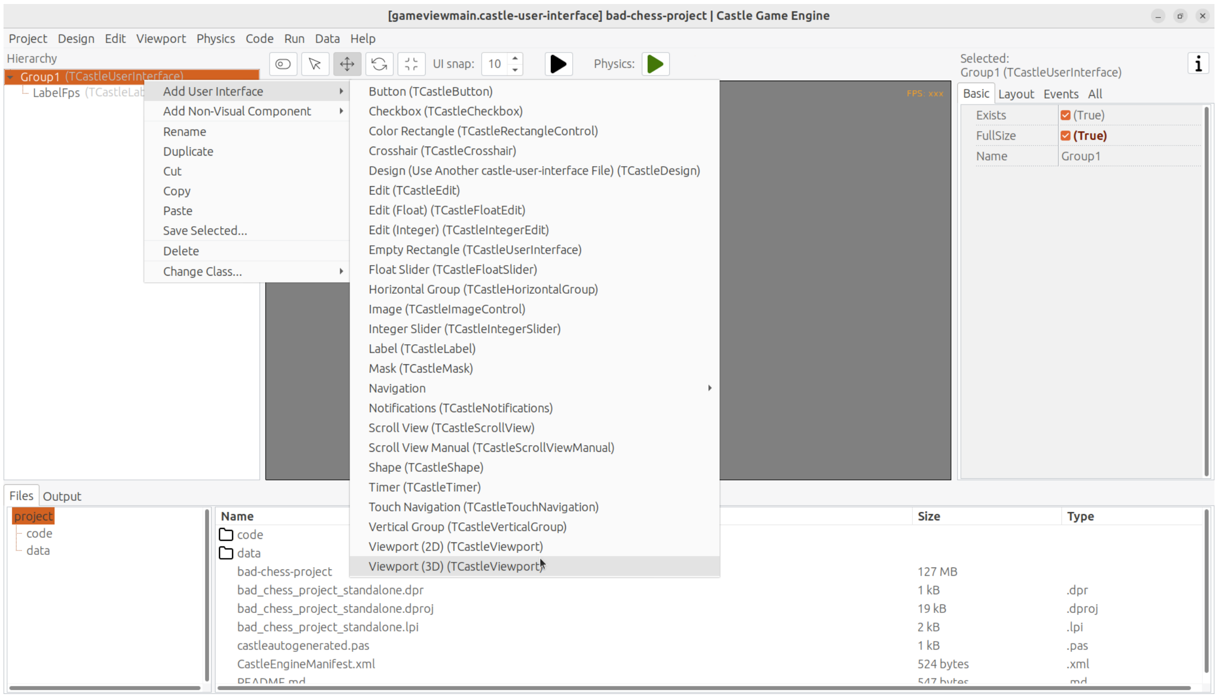
The initial view is mostly empty.

- It has a root component `Group1`, which is an instance of `TCastleUserInterface`. This component will contain everything else we design.
- And it has a label `LabelFps` (an instance of `TCastleLabel` class). At run-time, this label will display the FPS counter.


[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
Group1 (TCastleUserInterface)
LabelFps (TCastleLabel)
UI snap: 10
Physics:
Nothing Selected
FPS: xxx
Files Output
project
code
data

Name	Size	Type
code		
data		
bad-chess-project	127 MB	
bad_chess_project_standalone.dpr	1 kB	.dpr
bad_chess_project_standalone.dproj	19 kB	.dproj
bad_chess_project_standalone.lpi	2 kB	.lpi
castleautogenerated.pas	1 kB	.pas
CastleEngineManifest.xml	524 bytes	.xml
README.md	547 bytes	.md

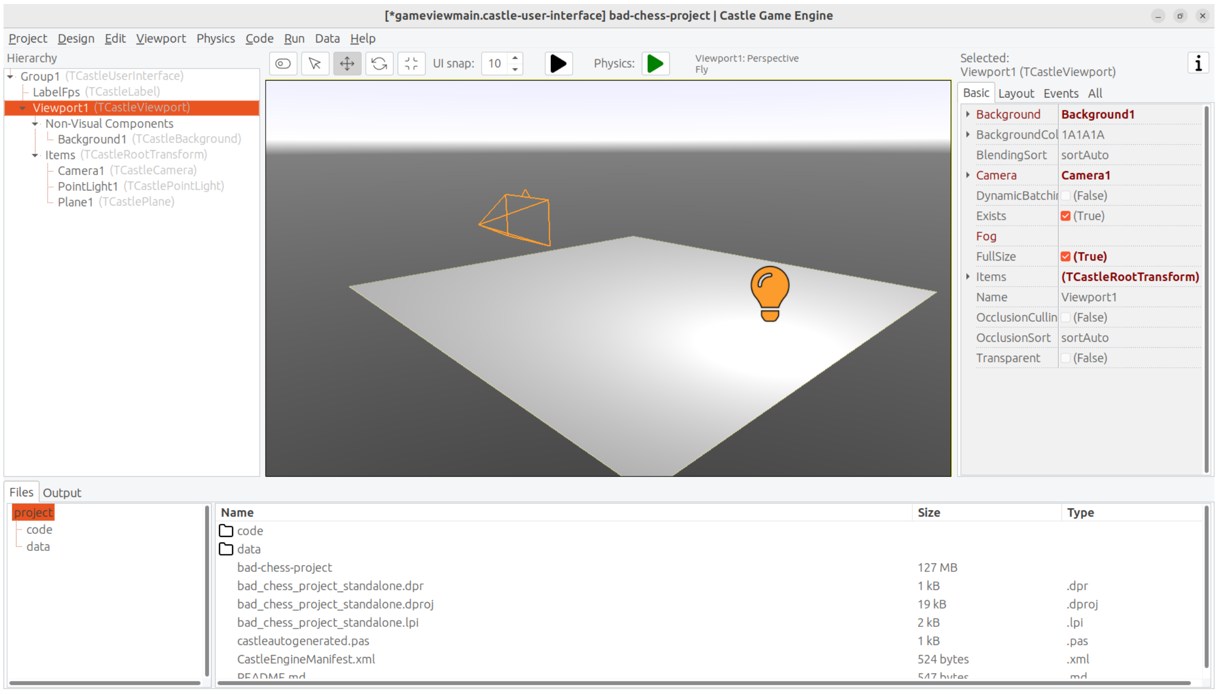
Let's add more content to it. First of all, to display anything in 3D, you need a *viewport*. A viewport is a way to display 3D or 2D content. It is an instance of `TCastleViewport` class. Add it to the design by right-clicking on the `Group1` component and choosing *"Add User Interface # Viewport (3D)"* from the menu that appears.


[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
Group1 (TCastleUserInterface)
LabelFps (TCastleLabel)
UI snap: 10
Physics:
Add User Interface
Add Non-Visual Component
Rename
Duplicate
Cut
Copy
Paste
Save Selected...
Delete
Change Class...
Button (TCastleButton)
Checkbox (TCastleCheckbox)
Color Rectangle (TCastleRectangleControl)
Crosshair (TCastleCrosshair)
Design (Use Another castle-user-interface File) (TCastleDesign)
Edit (TCastleEdit)
Edit (Float) (TCastleFloatEdit)
Edit (Integer) (TCastleIntegerEdit)
Empty Rectangle (TCastleUserInterface)
Float Slider (TCastleFloatSlider)
Horizontal Group (TCastleHorizontalGroup)
Image (TCastleImageControl)
Integer Slider (TCastleIntegerSlider)
Label (TCastleLabel)
Mask (TCastleMask)
Navigation
Notifications (TCastleNotifications)
Scroll View (TCastleScrollView)
Scroll View Manual (TCastleScrollViewManual)
Shape (TCastleShape)
Timer (TCastleTimer)
Touch Navigation (TCastleTouchNavigation)
Vertical Group (TCastleVerticalGroup)
Viewport (2D) (TCastleViewport)
Viewport (3D) (TCastleViewport)
Selected:
Group1 (TCastleUserInterface)
Basic Layout Events All
Files Output
project
code
data

Exists	(True)
FullSize	(True)
Name	Group1

Name	Size	Type
code		
data		
bad-chess-project	127 MB	
bad_chess_project_standalone.dpr	1 kB	.dpr
bad_chess_project_standalone.dproj	19 kB	.dproj
bad_chess_project_standalone.lpi	2 kB	.lpi
castleautogenerated.pas	1 kB	.pas
CastleEngineManifest.xml	524 bytes	.xml

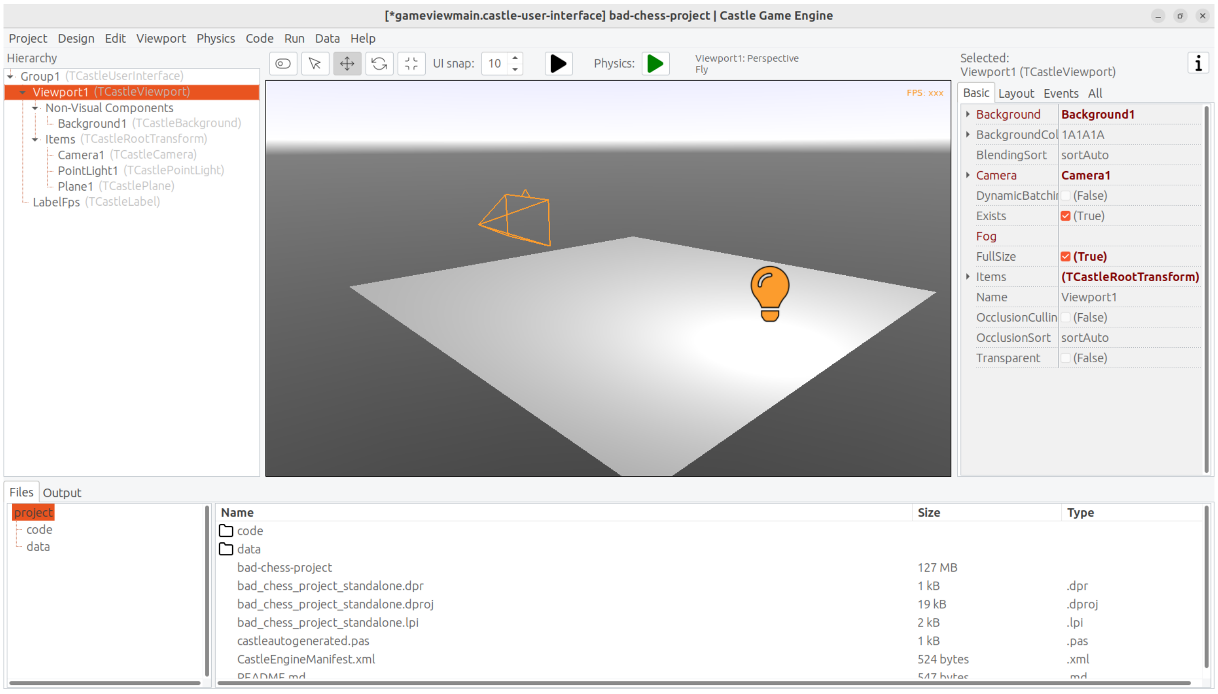
The result should look like this:


[*gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
Group1 (TCastleUserInterface)
LabelFps (TCastleLabel)
Viewport1 (TCastleViewport)
Non-Visual Components
Background1 (TCastleBackground)
Items (TCastleRootTransform)
Camera1 (TCastleCamera)
PointLight1 (TCastlePointLight)
Plane1 (TCastlePlane)
UI snap: 10
Physics:
Viewport1: Perspective
Fly
Selected:
Viewport1 (TCastleViewport)
Basic Layout Events All
Files Output
project
code
data

Background	Background1
BackgroundCol	1A1A1A
BlendingSort	sortAuto
Camera	Camera1
DynamicBatchi	(False)
Exists	(True)
Fog	
FullSize	(True)
Items	(TCastleRootTransform)
Name	Viewport1
OcclusionCullin	(False)
OcclusionSort	sortAuto
Transparent	(False)

Name	Size	Type
code		
data		
bad-chess-project	127 MB	
bad_chess_project_standalone.dpr	1 kB	.dpr
bad_chess_project_standalone.dproj	19 kB	.dproj
bad_chess_project_standalone.lpi	2 kB	.lpi
castleautogenerated.pas	1 kB	.pas
CastleEngineManifest.xml	524 bytes	.xml

Following this, drag the new `Viewport1` component above the `LabelFps` in the *Hierachy* panel (on the left). This way the FPS counter will be displayed in front of the viewport.


[*gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
UI snap: 10
Physics:
Viewport1: Perspective
Fly
FPS: xxx
Files Output

Group1 (TCastleUserInterface)
Viewport1 (TCastleViewport)
Non-Visual Components
Background1 (TCastleBackground)
Items (TCastleRootTransform)
Camera1 (TCastleCamera)
PointLight1 (TCastlePointLight)
Plane1 (TCastlePlane)
LabelFps (TCastleLabel)

Selected:
Viewport1 (TCastleViewport)

Basic Layout Events All

Background	Background1
BackgroundCol	1A1A1A
BlendingSort	sortAuto
Camera	Camera1
DynamicBatchi	(False)
Exists	✓ (True)
Fog	
FullSize	✓ (True)
Items	(TCastleRootTransform)
Name	Viewport1
OcclusionCullin	(False)
OcclusionSort	sortAuto
Transparent	(False)

project
code
data

Name	Size	Type
code		
data		
bad-chess-project	127 MB	
bad_chess_project_standalone.dpr	1 kB	.dpr
bad_chess_project_standalone.dproj	19 kB	.dproj
bad_chess_project_standalone.lpi	2 kB	.lpi
castleautogenerated.pas	1 kB	.pas
CastleEngineManifest.xml	524 bytes	.xml

Now play around in the 3D view. There are 3 objects in 3D world:

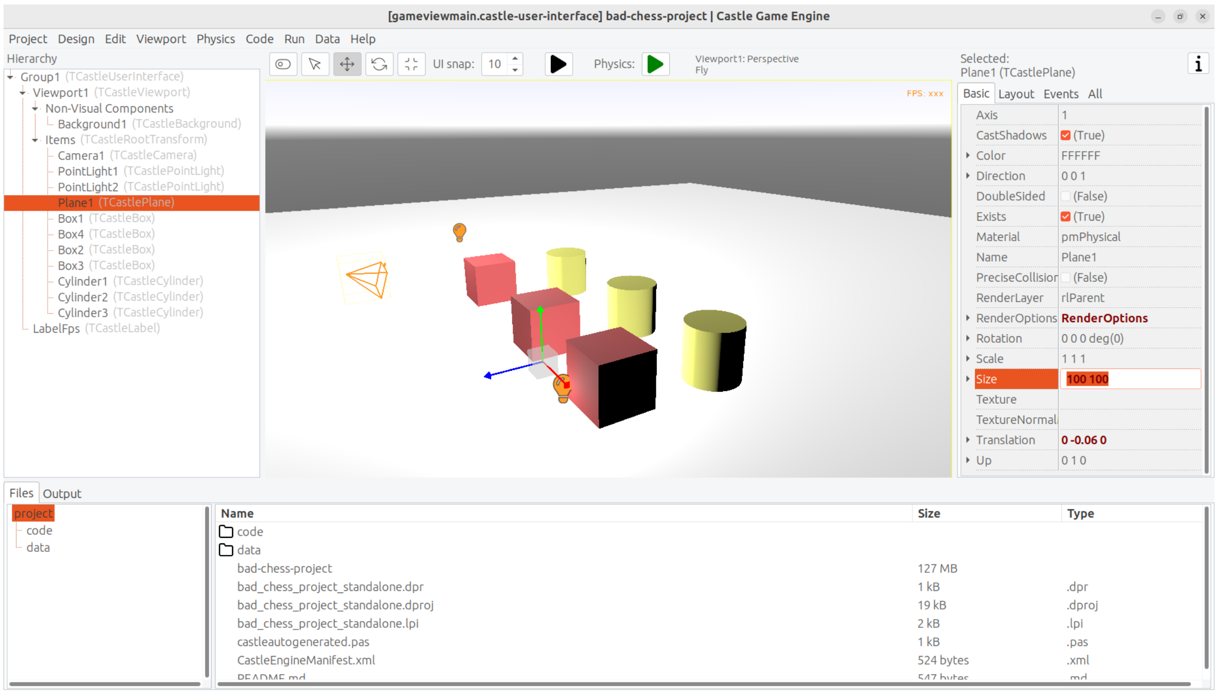
- Camera, called just `Camera1`, determines what the user will actually see once the game is run.
- Light source makes things *lit* (bright). The initial light source is called `PointLight1` and it is an instance of `TCastlePointLight`, which is a simple light that shines in all directions from a given 3D position.
- Rectangle representing a ground called a `Plane1`. Mathematically speaking, it's not a *plane*, it's a rectangle — however calling this a *"plane"* is a convention used by a lot of 3D software.

Click and hold the *right mouse button* over the viewport to look around. Use the AWSD keys to move. Use the *mouse scroll* (while holding the right mouse button pressed) to increase or decrease the movement speed.

Play around with **moving the items**. Drag the 3D axis to move any object.

Play around with **adding new 3D items**. Right-click on `Items` component inside the `Viewport1` and from the context menu add primitives like *"Box"*, *"Sphere"*, *"Cylinder"*. Move them around, delete them (with *Delete* key), duplicate (with *Ctrl+D* key).

Change some properties. On the right side, you can see an *object inspector*, familiar to any Lazarus and Delphi user. Adjust the properties, for example change the `Size` of the `Plane1` to be much bigger. Click on *"…"* (3 dots) button at the *"Color"* property of any primitive (like a plane, a box, a sphere…) to change the color.


[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
Group1 (TCastleUserInterface)
Viewport1 (TCastleViewport)
Non-Visual Components
Background1 (TCastleBackground)
Items (TCastleRootTransform)
Camera1 (TCastleCamera)
PointLight1 (TCastlePointLight)
PointLight2 (TCastlePointLight)
Plane1 (TCastlePlane)
Box1 (TCastleBox)
Box4 (TCastleBox)
Box2 (TCastleBox)
Box3 (TCastleBox)
Cylinder1 (TCastleCylinder)
Cylinder2 (TCastleCylinder)
Cylinder3 (TCastleCylinder)
LabelFps (TCastleLabel)
UI snap: 10
Physics:
Viewport1: Perspective
Fly
FPS: xxx
Files Output
project
code
data

Selected:
Plane1 (TCastlePlane)

Basic Layout Events All

Axis	1
CastShadows	✓ (True)
Color	FFFFFF
Direction	0 0 1
DoubleSided	(False)
Exists	✓ (True)
Material	pmPhysical
Name	Plane1
PreciseCollision	(False)
RenderLayer	rlParent
RenderOptions	**RenderOptions**
Rotation	0 0 0 deg(0)
Scale	1 1 1
Size	100 100
Texture	
TextureNormal	
Translation	**0 -0.06 0**
Up	0 1 0

Name	Size	Type
code		
data		
bad-chess-project	127 MB	
bad_chess_project_standalone.dpr	1 kB	.dpr
bad_chess_project_standalone.dproj	19 kB	.dproj
bad_chess_project_standalone.lpi	2 kB	.lpi
castleautogenerated.pas	1 kB	.pas
CastleEngineManifest.xml	524 bytes	.xml
README.md	547 bytes	.md

If you get stuck, consult our manual, in particular `https://castle-engine.io/viewport_and_scenes` and `https://castle-engine.io/viewport_3d` are helpful to learn basic 3D manipulation.

7. Design a 3D chessboard with chess pieces

Above we learned to design a 3D world composed from simple primitives, like boxes and spheres.

But this isn't a way to create realistic 3D graphics. In most 3D graphic applications, the content is created using a specialized 3D *authoring tool*, like *Blender*. 3D artist creates a *mesh* (a set of vertexes, connected to form edges and polygons), assigns materials and textures, and exports the resulting object to a file that can be read by a game engine — like a glTF file.

glTF is a full-featured 3D model format developed by Khronos. *Castle Game Engine* has great support for glTF, see `https://castle-engine.io/gltf` for details.

On *Castle Game Engine* side, our most important component to display a 3D model is `TCastleScene`. It's a big component, playing central role in our engine (in one way or another, it is actually responsible for all of 3D and 2D rendering in our viewport).

Using it is simple: you create an instance of `TCastleScene` and set its `URL` property to point to the model you want to display (like a glTF file). The `TCastleScene` class descends from the `TCastleTransform` class, and as such you can move, rotate and scale the `TCastleScene` instances. Alternatively, you can also drag-and-drop the glTF file from the *"Files"* panel to the viewport, editor will then automatically create a `TCastleScene` instance that loads the given model.

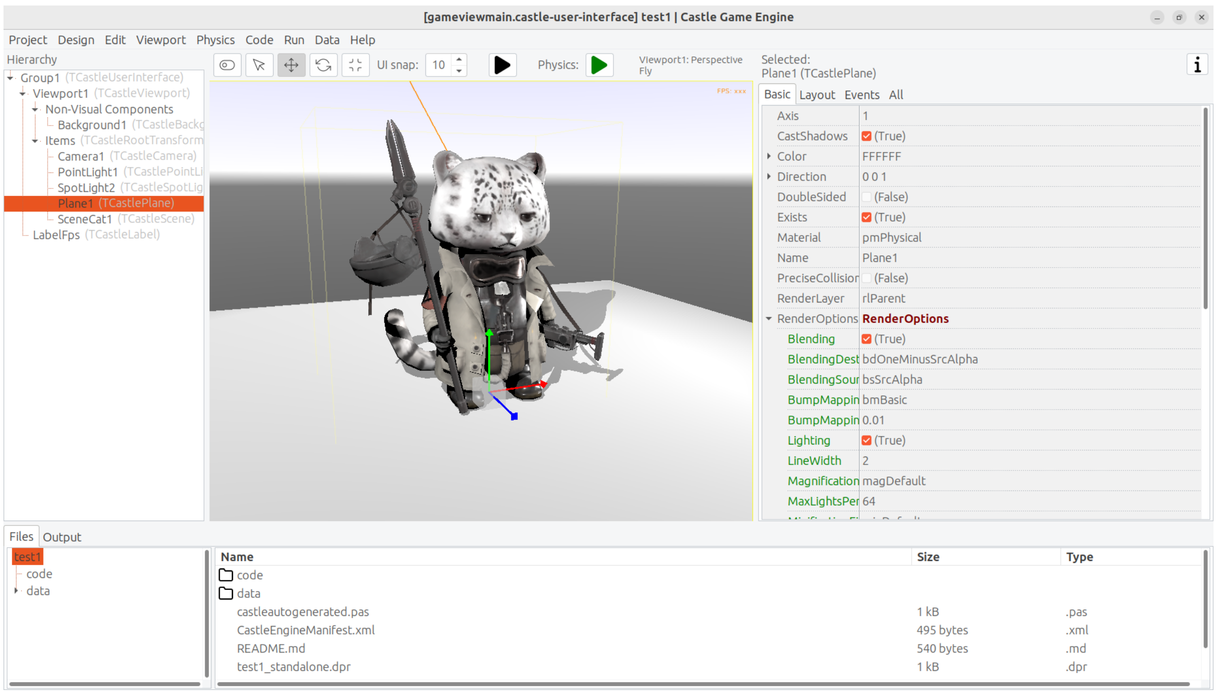
We support a number of 3D and 2D model formats, not only glTF. They are listed on `https://castle-engine.io/creating_data_model_formats.php` .

If you are capable of creating your own 3D models, for example in *Blender*, you can now make a detour: design a 3D model in Blender and export it to glTF using our instructions on `https://castle-engine.io/blender` .

Or you can use some ready-made stuff:

- There's a number of high-quality 3D content on the Internet, available also for free and on open-source-compatible licenses. We collect useful links on `https://castle-engine.io/assets.php` .
- Our engine also features an integration with *Sketchfab*, to allow you to search and download from a vast repository of free 3D models without leaving our editor. See the `https://castle-engine.io/sketchfab` documentation.

  Here's a sample — battle-hardened cat model, from Sketchfab, right inside our editor:



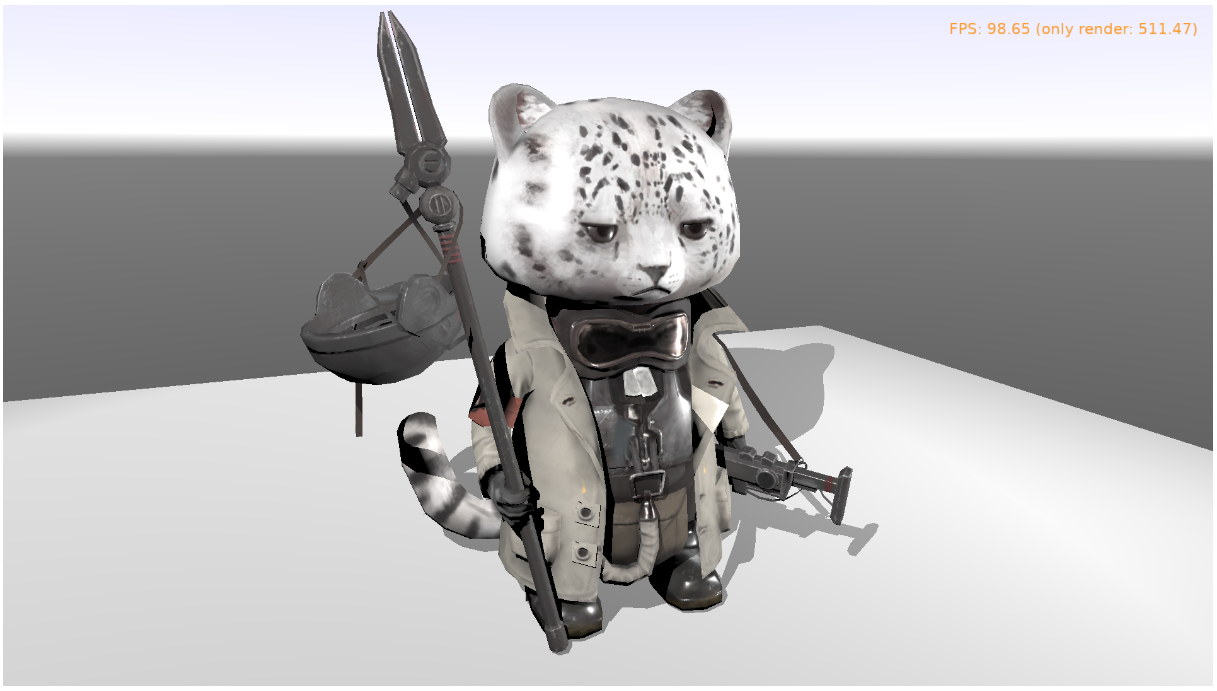
```
Group1 (TCastleUserInterface)
  Viewport1 (TCastleViewport)
    Non-Visual Components
      Background1 (TCastleBackg
    Items (TCastleRootTransform
      Camera1 (TCastleCamera)
      PointLight1 (TCastlePointLi
      SpotLight2 (TCastleSpotLig
      Plane1 (TCastlePlane)
      SceneCat1 (TCastleScene)
  LabelFps (TCastleLabel)
```

Selected:
Plane1 (TCastlePlane)

Axis	1
CastShadows	✓ (True)
Color	FFFFFF
Direction	0 0 1
DoubleSided	(False)
Exists	✓ (True)
Material	pmPhysical
Name	Plane1
PreciseCollisior	(False)
RenderLayer	rlParent
RenderOptions	RenderOptions
Blending	✓ (True)
BlendingDest	bdOneMinusSrcAlpha
BlendingSour	bsSrcAlpha
BumpMappin	bmBasic
BumpMappin	0.01
Lighting	✓ (True)
LineWidth	2
Magnification	magDefault
MaxLightsPer	64

```
test1
  code
  data
```

Name	Size	Type
code		
data		
castleautogenerated.pas	1 kB	.pas
CastleEngineManifest.xml	495 bytes	.xml
README.md	540 bytes	.md
test1_standalone.dpr	1 kB	.dpr



Credits: The "Cat" 3D model was done by Muru (`https://sketchfab.com/muru`) and is available on Sketchfab (`https://sketchfab.com/3d-models/cat-16c3444c8d1440fc97fdf10f60ec58b0`) on CC-BY-4.0 license.

- Finally, we have a ready set of 3D models for the chessboard and all chess pieces, that you can use for this demo.

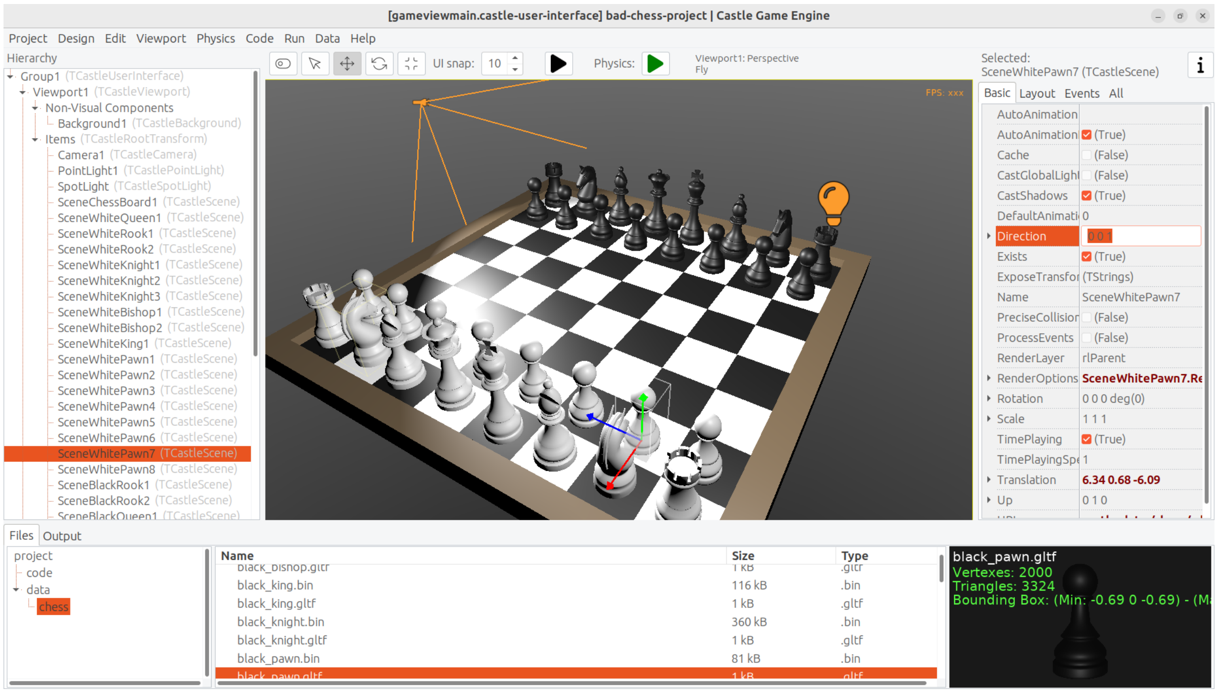
To use the last option, download the 3D models from `https://github.com/castle-engine/bad-chess/releases/download/chess-models/chess-models.zip` . They were made based on open-source Blender model published on `https://blendswap.com/blend/29244` by *Phuong2647*.

Unpack the resulting archive anywhere *under the `data` subdirectory of your project*.

Then simply drag-and-drop the `*.gltf` files onto the viewport. Move and duplicate them as needed, to arrange them into a starting chess position.

For our silly physics game, it actually completely doesn't matter how you will arrange them. You also don't need to position and rotate them perfectly. Have fun :)

This is an example result:


[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
UI snap: 10
Physics:
Viewport1: Perspective
Fly
FPS: xxx
Basic Layout Events All
Files Output

Group1 (TCastleUserInterface)
Viewport1 (TCastleViewport)
Non-Visual Components
Background1 (TCastleBackground)
Items (TCastleRootTransform)
Camera1 (TCastleCamera)
PointLight1 (TCastlePointLight)
SpotLight (TCastleSpotLight)
SceneChessBoard1 (TCastleScene)
SceneWhiteQueen1 (TCastleScene)
SceneWhiteRook1 (TCastleScene)
SceneWhiteRook2 (TCastleScene)
SceneWhiteKnight1 (TCastleScene)
SceneWhiteKnight2 (TCastleScene)
SceneWhiteKnight3 (TCastleScene)
SceneWhiteBishop1 (TCastleScene)
SceneWhiteBishop2 (TCastleScene)
SceneWhiteKing1 (TCastleScene)
SceneWhitePawn1 (TCastleScene)
SceneWhitePawn2 (TCastleScene)
SceneWhitePawn3 (TCastleScene)
SceneWhitePawn4 (TCastleScene)
SceneWhitePawn5 (TCastleScene)
SceneWhitePawn6 (TCastleScene)
SceneWhitePawn7 (TCastleScene)
SceneWhitePawn8 (TCastleScene)
SceneBlackRook1 (TCastleScene)
SceneBlackRook2 (TCastleScene)
SceneBlackQueen1 (TCastleScene)

Selected:
SceneWhitePawn7 (TCastleScene)

AutoAnimation	
AutoAnimation	✓ (True)
Cache	(False)
CastGlobalLigh	(False)
CastShadows	✓ (True)
DefaultAnimati	0
Direction	0 0 1
Exists	✓ (True)
ExposeTransfor	(TStrings)
Name	SceneWhitePawn7
PreciseCollisior	(False)
ProcessEvents	(False)
RenderLayer	rlParent
RenderOptions	SceneWhitePawn7.Re
Rotation	0 0 0 deg(0)
Scale	1 1 1
TimePlaying	✓ (True)
TimePlayingSpe	1
Translation	6.34 0.68 -6.09
Up	0 1 0

project
code
data
chess

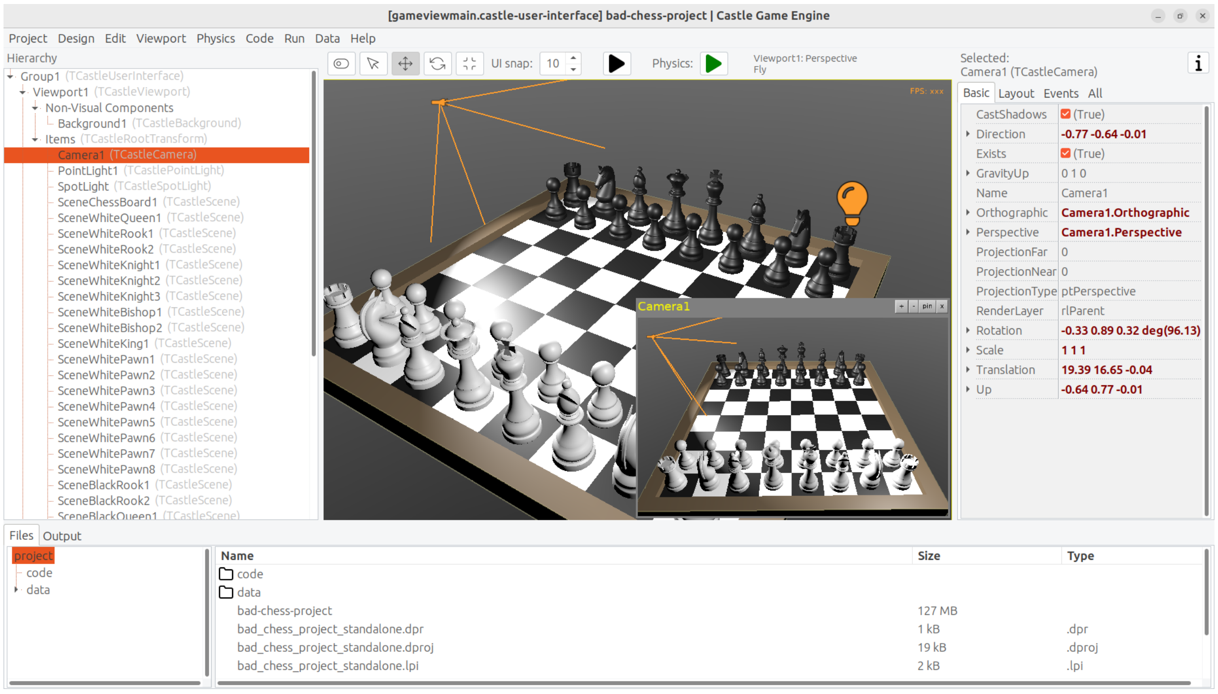
Name	Size	Type
black_bishop.gltf	1 kB	.gltf
black_king.bin	116 kB	.bin
black_king.gltf	1 kB	.gltf
black_knight.bin	360 kB	.bin
black_knight.gltf	1 kB	.gltf
black_pawn.bin	81 kB	.bin
black_pawn.gltf	1 kB	.gltf

black_pawn.gltf
Vertexes: 2000
Triangles: 3324
Bounding Box: (Min: -0.69 0 -0.69) - (M

Once you've designed the chessboard and put chess pieces on it, also make sure to adjust the lights to make everything nicely bright (but not too bright).

Finally, adjust the camera so that user sees a nice view of the board when the application starts. When you select a camera component (like `Camera1`, if you haven't renamed the default camera), the editor shows a small window with *camera preview*. You can click *"Pin"* in this window to keep observing the world from this camera. There are basically 2 ways to manipulate the camera:

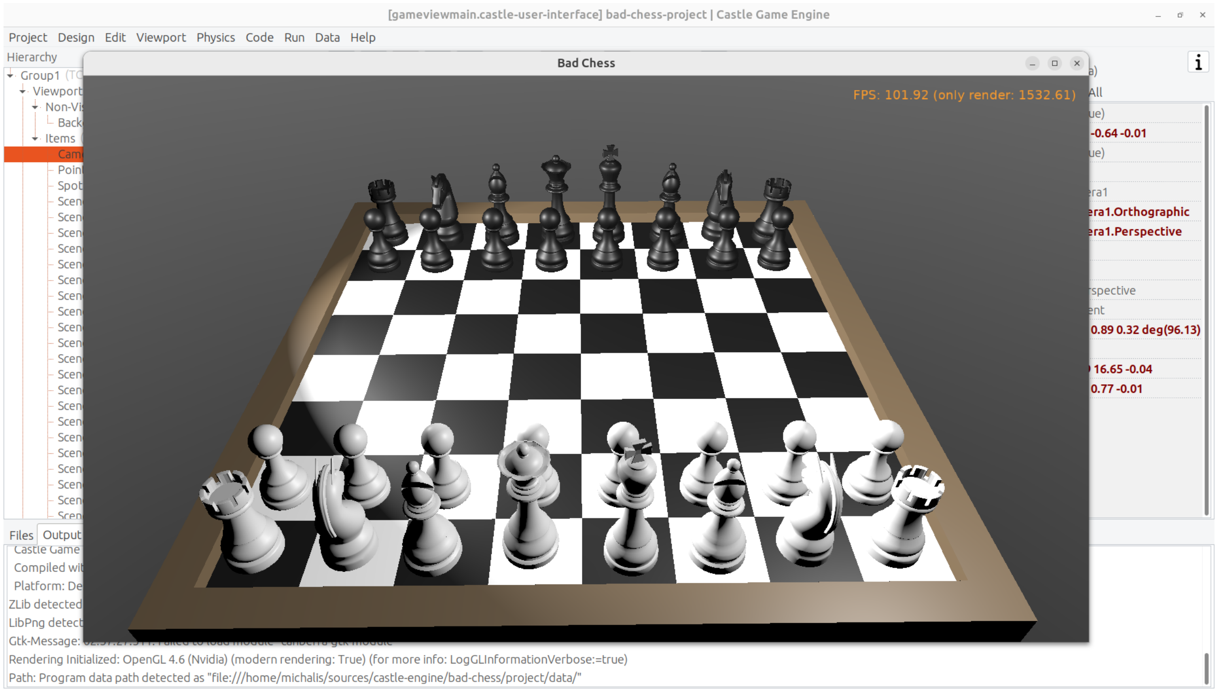
1. Move and rotate the camera just like any other 3D object. Look at the *camera preview* to judge whether the camera view looks good.
2. Or, alternatively, navigate in the editor and then use the menu item *"Viewport # Align Camera To View"* (key shortcut *Ctrl + Numpad 0*) to make the camera view match the current view in the editor.


[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
Group1 (TCastleUserInterface)
Viewport1 (TCastleViewport)
Non-Visual Components
Background1 (TCastleBackground)
Items (TCastleRootTransform)
Camera1 (TCastleCamera)
PointLight1 (TCastlePointLight)
SpotLight (TCastleSpotLight)
SceneChessBoard1 (TCastleScene)
SceneWhiteQueen1 (TCastleScene)
SceneWhiteRook1 (TCastleScene)
SceneWhiteRook2 (TCastleScene)
SceneWhiteKnight1 (TCastleScene)
SceneWhiteKnight2 (TCastleScene)
SceneWhiteKnight3 (TCastleScene)
SceneWhiteBishop1 (TCastleScene)
SceneWhiteBishop2 (TCastleScene)
SceneWhiteKing1 (TCastleScene)
SceneWhitePawn1 (TCastleScene)
SceneWhitePawn2 (TCastleScene)
SceneWhitePawn3 (TCastleScene)
SceneWhitePawn4 (TCastleScene)
SceneWhitePawn5 (TCastleScene)
SceneWhitePawn6 (TCastleScene)
SceneWhitePawn7 (TCastleScene)
SceneWhitePawn8 (TCastleScene)
SceneBlackRook1 (TCastleScene)
SceneBlackRook2 (TCastleScene)
SceneBlackQueen1 (TCastleScene)
UI snap: 10
Physics:
Viewport1: Perspective
Fly
FPS: xxx
Camera1
Selected:
Camera1 (TCastleCamera)
Basic Layout Events All
Files Output
project
code
data

CastShadows	✓ (True)
Direction	-0.77 -0.64 -0.01
Exists	✓ (True)
GravityUp	0 1 0
Name	Camera1
Orthographic	Camera1.Orthographic
Perspective	Camera1.Perspective
ProjectionFar	0
ProjectionNear	0
ProjectionType	ptPerspective
RenderLayer	rlParent
Rotation	-0.33 0.89 0.32 deg(96.13)
Scale	1 1 1
Translation	19.39 16.65 -0.04
Up	-0.64 0.77 -0.01

Name	Size	Type
code		
data		
bad-chess-project	127 MB	
bad_chess_project_standalone.dpr	1 kB	.dpr
bad_chess_project_standalone.dproj	19 kB	.dproj
bad_chess_project_standalone.lpi	2 kB	.lpi

Once you have a nice view, make sure it all works: compile and run the application again.


[gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine
Project Design Edit Viewport Physics Code Run Data Help
Hierarchy
Bad Chess
FPS: 101.92 (only render: 1532.61)
Files Output

```
Castle Game
Compiled wit
Platform: De
ZLib detected
LibPng detect
Gtk-Message:
Rendering Initialized: OpenGL 4.6 (Nvidia) (modern rendering: True) (for more info: LogGLInformationVerbose:=true)
Path: Program data path detected as "file:///home/michalis/sources/castle-engine/bad-chess/project/data/"
```

8. Using physics in the editor

Now that the proper chessboard with chess pieces is designed, let's use physics to make things *crazier*.

Castle Game Engine has a support for *rigid body physics*. This means that:

- Objects can be affected by *forces*.

  The force that works automatically is *gravity*, pulling objects down (in the direction of the negative Y axis, by default).

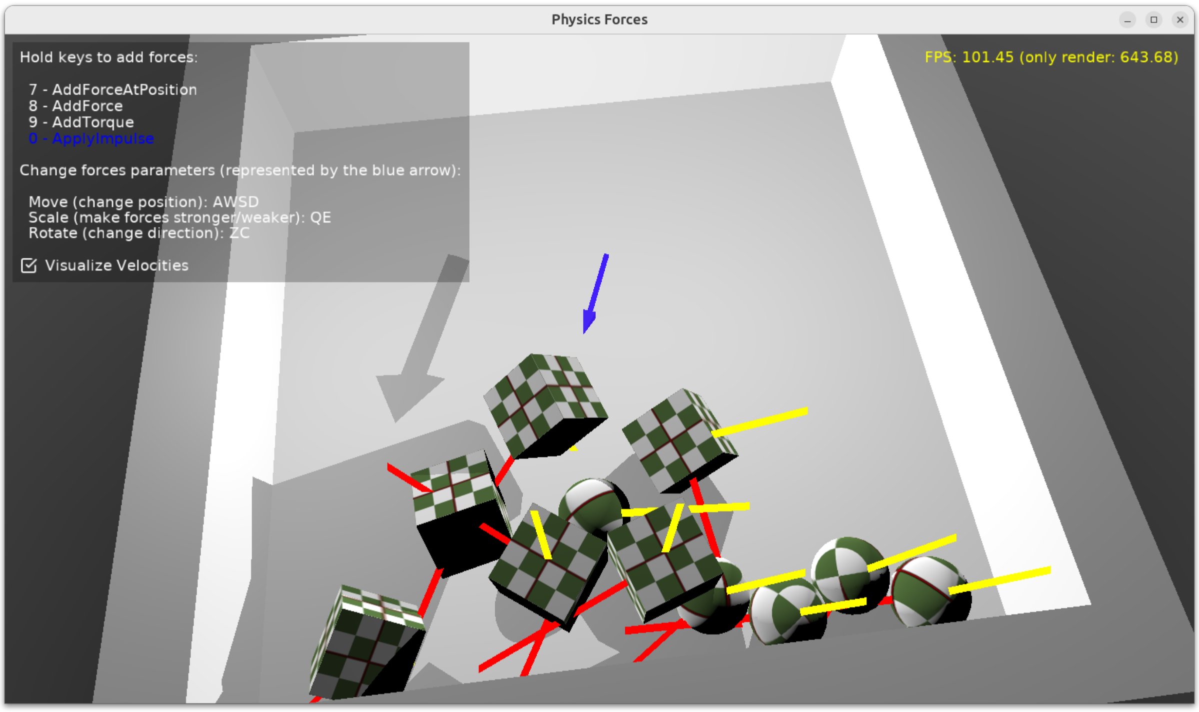
  You can also define additional forces from code, to e.g. push things along an arbitary direction. Your own forces can realize a range of real-life effects, like wind, explosions, spinning tornadoes, etc.

- Collisions between objects are automatically detected and resolved. That is, by default the objects will *bounce off* each other.

  It is also possible to detect collisions in code and react to them in any way (e.g. an enemy may explode when it collides with a rocket).

- You can also connect certain objects using *joints*.

We will not explore all these features in our article, but we will show you how to enjoy the basics. To learn more about the possibilities, read our manual `https://castle-engine.io/physics` and play with demoes in the `examples/physics/` subdirectory of the engine. Here's a screenshot from one of the demos, showing explicit application of physics forces:



Hold keys to add forces:

7 - AddForceAtPosition
8 - AddForce
9 - AddTorque
0 - ApplyImpulse

Change forces parameters (represented by the blue arrow):

Move (change position): AWSD
Scale (make forces stronger/weaker): QE
Rotate (change direction): ZC

☑ Visualize Velocities

FPS: 101.45 (only render: 643.68)

Castle Game Engine physics internally uses *Kraft*, a physics engine deveveloped in Pascal by *Benjamin 'BeRo' Rosseaux*.

Any component descending `TCastleTransform`, including primitives (like `TCastleBox`) or scenes loaded from models (`TCastleScene`) or a group of other objects (`TCastleTransform` with children) can be a *rigid body* for the physics engine that participates in the collision detection and resulting movement. The object needs to have two *behaviors*:

1. `TCastleRigidBody` behavior makes the component a *rigid body*. It defines common physics properties, like whether the object is affected by gravity and the initial movement speed.

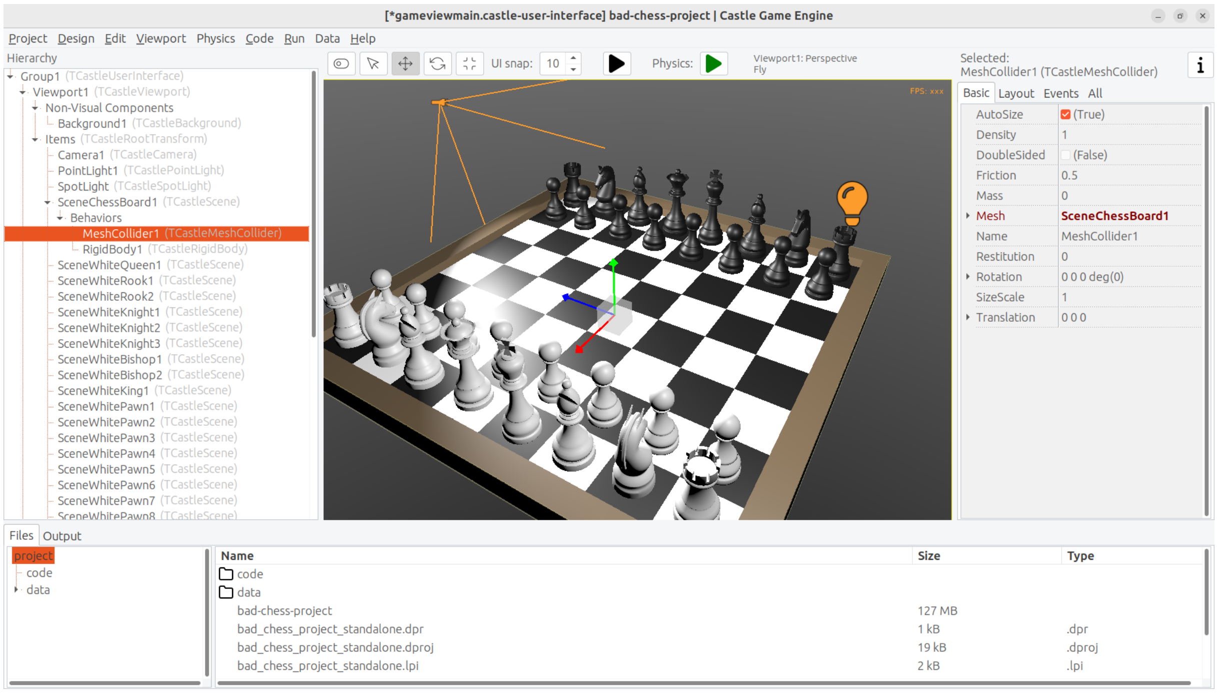
2. A *collider*, which stands for *any component descending from the abstract class `TCastleCollider`.* Many collider shapes are possible, like `TCastleSphereCollider`, `TCastleBoxCollider` and `TCastleMeshCollider`.

   Using the `TCastleMeshCollider` results in most precise collisions, but the colliding object must be *static* which means that other objects will *bounce off* this object, but the object with `TCastleMeshCollider` will not move itself.

The term *behavior* we used above is a special mechanism in *Castle Game Engine* to attach additional functionality to a `TCastleTransform`. Behaviors are a great way to define various functionality that enhances given game object. There are various built-in behaviors and you can also define your own. See `https://castle-engine.io/behaviors` for more information.

After this overview, you're ready to actually use physics in our chess game.

Right-click on the component representing the chessboard. From the context menu choose *"Add Behavior (Extends Parent Transform) # Physics # Collider # Mesh"*. In response, you will notice that 2 components have appeared in the component tree: `MeshCollider1` and `RigidBody1`. That's a convenience feature of the editor: adding a collider also adds a rigid body component.



[*gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine

Group1 (TCastleUserInterface)
Viewport1 (TCastleViewport)
Non-Visual Components
Background1 (TCastleBackground)
Items (TCastleRootTransform)
Camera1 (TCastleCamera)
PointLight1 (TCastlePointLight)
SpotLight (TCastleSpotLight)
SceneChessBoard1 (TCastleScene)
Behaviors
MeshCollider1 (TCastleMeshCollider)
RigidBody1 (TCastleRigidBody)
SceneWhiteQueen1 (TCastleScene)
SceneWhiteRook1 (TCastleScene)
SceneWhiteRook2 (TCastleScene)
SceneWhiteKnight1 (TCastleScene)
SceneWhiteKnight2 (TCastleScene)
SceneWhiteKnight3 (TCastleScene)
SceneWhiteBishop1 (TCastleScene)
SceneWhiteBishop2 (TCastleScene)
SceneWhiteKing1 (TCastleScene)
SceneWhitePawn1 (TCastleScene)
SceneWhitePawn2 (TCastleScene)
SceneWhitePawn3 (TCastleScene)
SceneWhitePawn4 (TCastleScene)
SceneWhitePawn5 (TCastleScene)
SceneWhitePawn6 (TCastleScene)
SceneWhitePawn7 (TCastleScene)
SceneWhitePawn8 (TCastleScene)

Selected:
MeshCollider1 (TCastleMeshCollider)

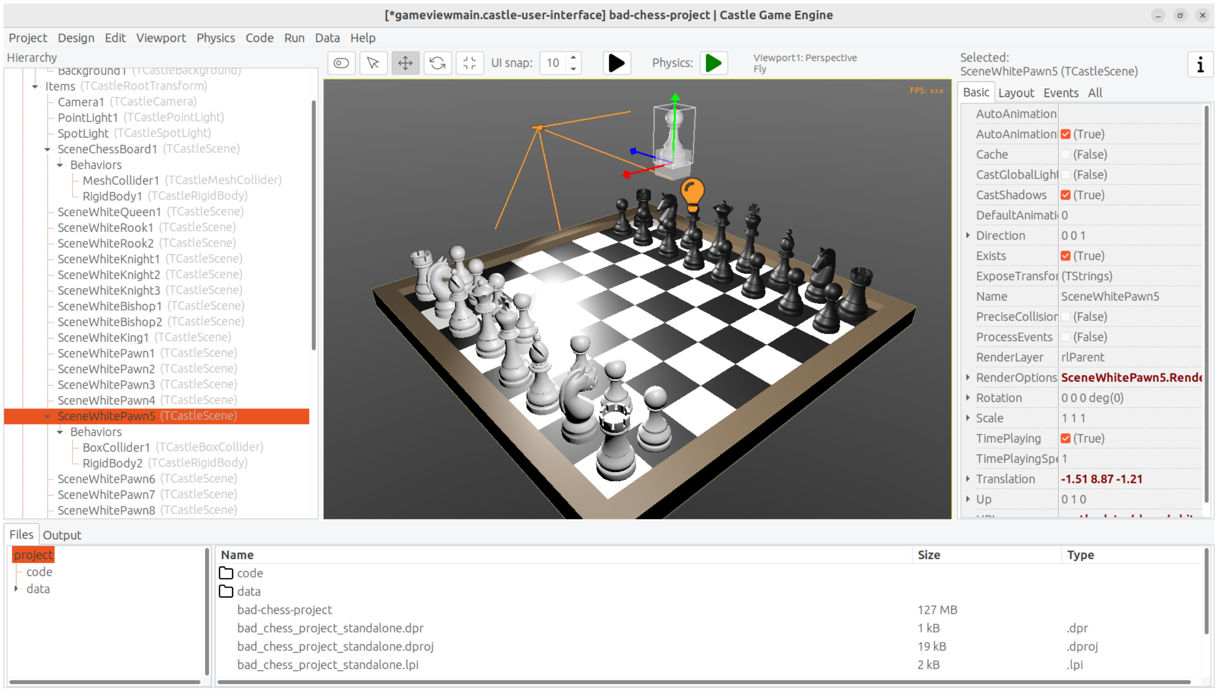
AutoSize	✓ (True)
Density	1
DoubleSided	(False)
Friction	0.5
Mass	0
Mesh	SceneChessBoard1
Name	MeshCollider1
Restitution	0
Rotation	0 0 0 deg(0)
SizeScale	1
Translation	0 0 0

project
code
data

Name	Size	Type
code		
data		
bad-chess-project	127 MB	
bad_chess_project_standalone.dpr	1 kB	.dpr
bad_chess_project_standalone.dproj	19 kB	.dproj
bad_chess_project_standalone.lpi	2 kB	.lpi

Next choose any chess piece. Right-click on it and from the context menu choose *"Add Behavior (Extends Parent Transform) # Physics # Collider # Box"*. Note that we use a simpler collider for the chess piece, which is also dynamic. This will allow the chess piece to actually fall down on the board.

Finally move the chess piece to a more dramatic position, above the board, so that it will fall down when the physics will start.



[*gameviewmain.castle-user-interface] bad-chess-project | Castle Game Engine

Background1 (TCastleBackground)
Items (TCastleRootTransform)
Camera1 (TCastleCamera)
PointLight1 (TCastlePointLight)
SpotLight (TCastleSpotLight)
SceneChessBoard1 (TCastleScene)
Behaviors
MeshCollider1 (TCastleMeshCollider)
RigidBody1 (TCastleRigidBody)
SceneWhiteQueen1 (TCastleScene)
SceneWhiteRook1 (TCastleScene)
SceneWhiteRook2 (TCastleScene)
SceneWhiteKnight1 (TCastleScene)
SceneWhiteKnight2 (TCastleScene)
SceneWhiteKnight3 (TCastleScene)
SceneWhiteBishop1 (TCastleScene)
SceneWhiteBishop2 (TCastleScene)
SceneWhiteKing1 (TCastleScene)
SceneWhitePawn1 (TCastleScene)
SceneWhitePawn2 (TCastleScene)
SceneWhitePawn3 (TCastleScene)
SceneWhitePawn4 (TCastleScene)
SceneWhitePawn5 (TCastleScene)
Behaviors
BoxCollider1 (TCastleBoxCollider)
RigidBody2 (TCastleRigidBody)
SceneWhitePawn6 (TCastleScene)
SceneWhitePawn7 (TCastleScene)
SceneWhitePawn8 (TCastleScene)

Selected:
SceneWhitePawn5 (TCastleScene)

AutoAnimation	
AutoAnimation	✓ (True)
Cache	(False)
CastGlobalLight	(False)
CastShadows	✓ (True)
DefaultAnimati	0
Direction	0 0 1
Exists	✓ (True)
ExposeTransfo	(TStrings)
Name	SceneWhitePawn5
PreciseCollisio	(False)
ProcessEvents	(False)
RenderLayer	rlParent
RenderOptions	SceneWhitePawn5.Rende
Rotation	0 0 0 deg(0)
Scale	1 1 1
TimePlaying	✓ (True)
TimePlayingSp	1
Translation	-1.51 8.87 -1.21
Up	0 1 0

project
code
data

Name	Size	Type
code		
data		
bad-chess-project	127 MB	
bad_chess_project_standalone.dpr	1 kB	.dpr
bad_chess_project_standalone.dproj	19 kB	.dproj
bad_chess_project_standalone.lpi	2 kB	.lpi

We are ready to run physics. One way would be to just run the application, using the *"Compile And Run"* as you’ve done before. But there’s a quicker way to experiment with physics: run *physics simulation* by using the green play icon at the header of the editor (or menu item *"Physics # Play Simulation"*, key shortcut *Ctrl+P*).

Do this and watch in awe as the pawn falls on the board.

Remember to finish the physics simulation when you’re done (press the green stop button, or again menu item *"Physics # Play Simulation"*, key shortcut *Ctrl+P*). Editing the design during the physics simulation is allowed (and it’s a great way to experiment with various physics settings) but *the changes are not saved when physics simulation is running*. That’s because physics typically moves the objects, and you don’t want to save this position resulting from physics interactions. So be sure to *stop the physics simulation before doing any persistent changes to the design*.

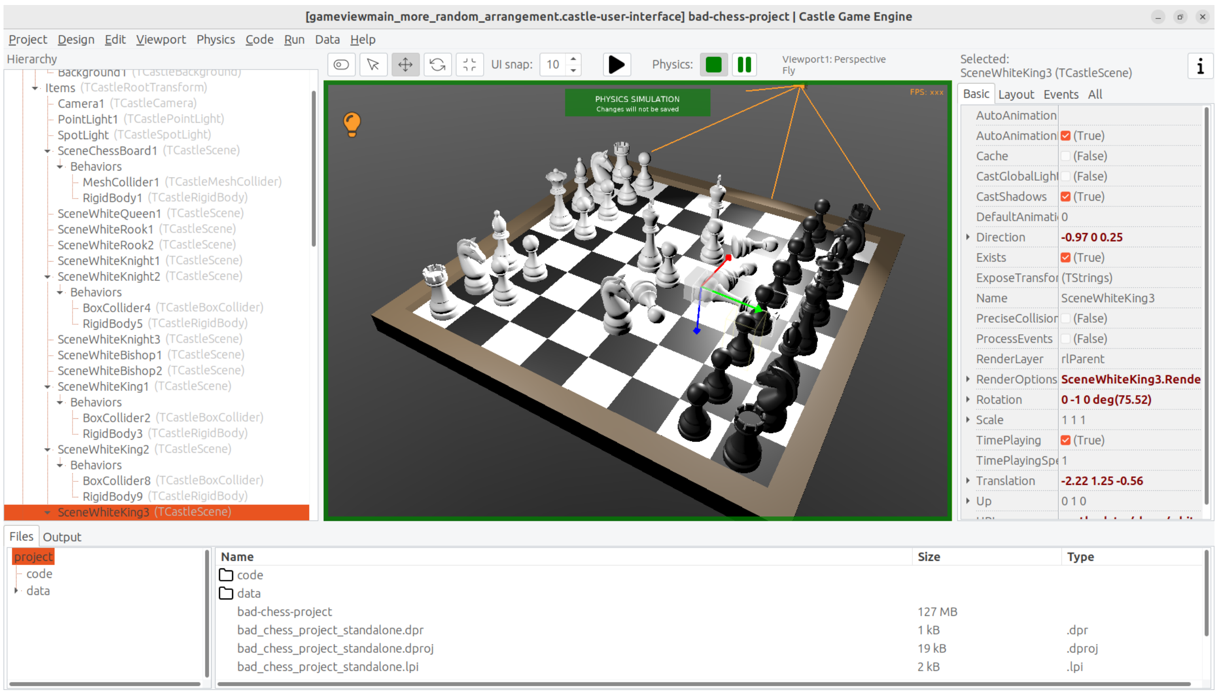
To get more spectacular results:

- Add physics colliders to more chess pieces.
- Move the chess pieces to more interesting positions, so that multiple pieces will fall down from above on multiple other chess pieces.
- You can also duplicate (key shortcut *Ctrl+D*) the chess pieces (it will duplicate the whole selected object, including physics behaviors if any). That s an easy way to have a lot of physical objects that bounce off each other.

After each change, just play and stop physics simulation again.

Make sure that the initial position of all rigid bodies does not make some pair collide with each other right at the start. If the two objects will collide at start, physics engine may (sometimes quite explosively) move them away from each other.

This is a sample result:



[gameviewmain_more_random_arrangement.castle-user-interface] bad-chess-project | Castle Game Engine

Hierarchy

```
    Background1 (TCastleBackground)
  Items (TCastleRootTransform)
    Camera1 (TCastleCamera)
    PointLight1 (TCastlePointLight)
    SpotLight (TCastleSpotLight)
    SceneChessBoard1 (TCastleScene)
      Behaviors
        MeshCollider1 (TCastleMeshCollider)
        RigidBody1 (TCastleRigidBody)
    SceneWhiteQueen1 (TCastleScene)
    SceneWhiteRook1 (TCastleScene)
    SceneWhiteRook2 (TCastleScene)
    SceneWhiteKnight1 (TCastleScene)
    SceneWhiteKnight2 (TCastleScene)
      Behaviors
        BoxCollider4 (TCastleBoxCollider)
        RigidBody5 (TCastleRigidBody)
    SceneWhiteKnight3 (TCastleScene)
    SceneWhiteBishop1 (TCastleScene)
    SceneWhiteBishop2 (TCastleScene)
    SceneWhiteKing1 (TCastleScene)
      Behaviors
        BoxCollider2 (TCastleBoxCollider)
        RigidBody3 (TCastleRigidBody)
    SceneWhiteKing2 (TCastleScene)
      Behaviors
        BoxCollider8 (TCastleBoxCollider)
        RigidBody9 (TCastleRigidBody)
    SceneWhiteKing3 (TCastleScene)
```

Selected:
SceneWhiteKing3 (TCastleScene)

AutoAnimation	
AutoAnimation	✓ (True)
Cache	(False)
CastGlobalLigh	(False)
CastShadows	✓ (True)
DefaultAnimati	0
Direction	-0.97 0 0.25
Exists	✓ (True)
ExposeTransfor	(TStrings)
Name	SceneWhiteKing3
PreciseCollisior	(False)
ProcessEvents	(False)
RenderLayer	rlParent
RenderOptions	SceneWhiteKing3.Rende
Rotation	0 -1 0 deg(75.52)
Scale	1 1 1
TimePlaying	✓ (True)
TimePlayingSpe	1
Translation	-2.22 1.25 -0.56
Up	0 1 0

project
code
data

Name	Size	Type
code		
data		
bad-chess-project	127 MB	
bad_chess_project_standalone.dpr	1 kB	.dpr
bad_chess_project_standalone.dproj	19 kB	.dproj
bad_chess_project_standalone.lpi	2 kB	.lpi

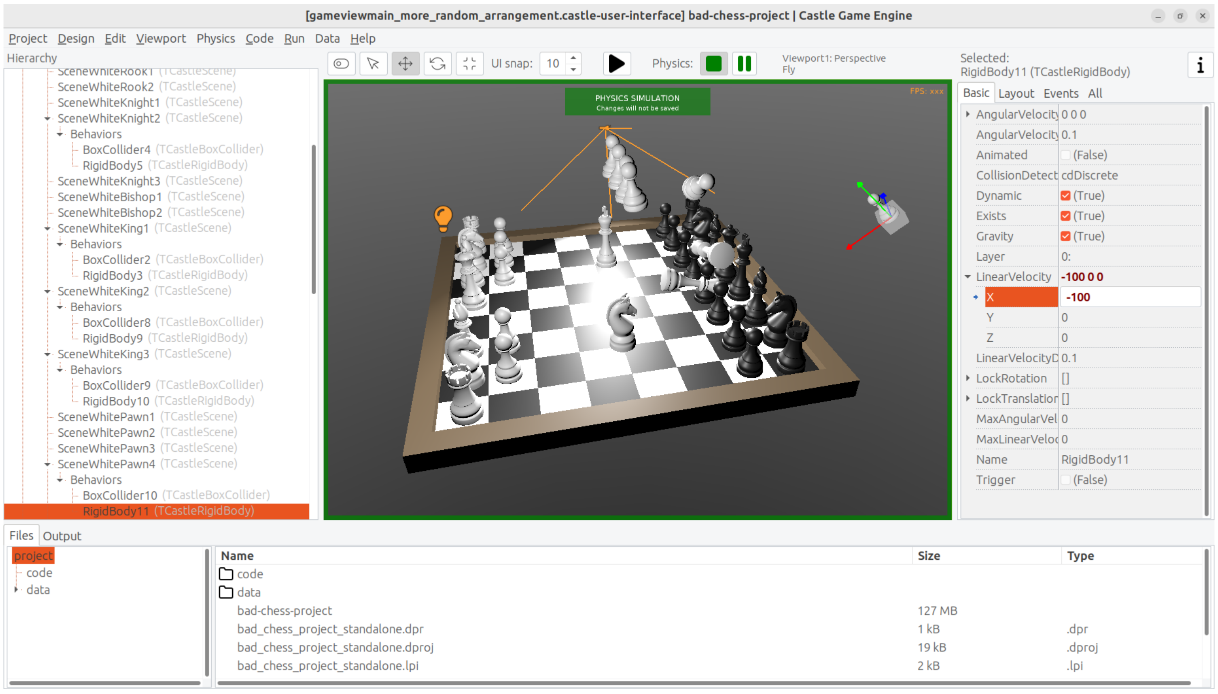
One last thing remains to learn in this (first) part of the article: how to *flick* the chess piece?

1. From Pascal code you can use various methods to apply a force on a rigid body. More about this in the next article part. You can also experiment with the example application `examples/physics/physics_forces/` if you're impatient.
2. Or you can set a specific `LinearVelocity` on a rigid body component.

We will use the latter approach, as it can be trivially done and tested in the editor.

- Select the chess piece. Any chess piece you want to *"flick"* (throw across the board).
- Make sure it has a collider and rigid body components (if not, add them, as above).
- Select the `TCastleRigidBody` component of it, and find the `LinearVelocity` property in it.
- Set `LinearVelocity` to any large non-zero vector, like `-100 0 0`. This means we have a velocity of 100 units per second in the negative X direction.

Run the physics simulation and watch the mayhem.



[gameviewmain_more_random_arrangement.castle-user-interface] bad-chess-project | Castle Game Engine

```
SceneWhiteRook1 (TCastleScene)
SceneWhiteRook2 (TCastleScene)
SceneWhiteKnight1 (TCastleScene)
SceneWhiteKnight2 (TCastleScene)
  Behaviors
    BoxCollider4 (TCastleBoxCollider)
    RigidBody5 (TCastleRigidBody)
SceneWhiteKnight3 (TCastleScene)
SceneWhiteBishop1 (TCastleScene)
SceneWhiteBishop2 (TCastleScene)
SceneWhiteKing1 (TCastleScene)
  Behaviors
    BoxCollider2 (TCastleBoxCollider)
    RigidBody3 (TCastleRigidBody)
SceneWhiteKing2 (TCastleScene)
  Behaviors
    BoxCollider8 (TCastleBoxCollider)
    RigidBody9 (TCastleRigidBody)
SceneWhiteKing3 (TCastleScene)
  Behaviors
    BoxCollider9 (TCastleBoxCollider)
    RigidBody10 (TCastleRigidBody)
SceneWhitePawn1 (TCastleScene)
SceneWhitePawn2 (TCastleScene)
SceneWhitePawn3 (TCastleScene)
SceneWhitePawn4 (TCastleScene)
  Behaviors
    BoxCollider10 (TCastleBoxCollider)
    RigidBody11 (TCastleRigidBody)
```

Selected:
RigidBody11 (TCastleRigidBody)

Basic Layout Events All

AngularVelocity	0 0 0
AngularVelocity	0.1
Animated	(False)
CollisionDetect	cdDiscrete
Dynamic	✓ (True)
Exists	✓ (True)
Gravity	✓ (True)
Layer	0:
LinearVelocity	-100 0 0
X	-100
Y	0
Z	0
LinearVelocityD	0.1
LockRotation	[]
LockTranslation	[]
MaxAngularVel	0
MaxLinearVeloc	0
Name	RigidBody11
Trigger	(False)

project
code
data

Name	Size	Type
code		
data		
bad-chess-project	127 MB	
bad_chess_project_standalone.dpr	1 kB	.dpr
bad_chess_project_standalone.dproj	19 kB	.dproj
bad_chess_project_standalone.lpi	2 kB	.lpi

9. Summary

We have designed a 3D application using *Castle Game Engine* with a bit of physics. We didn't yet write any Pascal code to do any interactions — this will be done in the next part of the article.

If you want to download a ready application, resulting from this, go to `https://github.com/castle-engine/bad-chess` . The subdirectory `project` of that repository contains the final working demo of this. It will be extended in the next part of the article.

I hope you had fun doing this demo and exploring the possibilities of *Castle Game Engine*.

If you have any questions or feedback about the engine, don't be shy! Speak up, ask and share your comments on our forum `https://forum.castle-engine.io` or Discord `https://castle-engine.io/talk.php` .