Castle Game Engine Tutorial

Michalis Kamburelis
michalis.kambi@gmail.com

Heraklion, Crete, Greece 18 - 21 June 2015

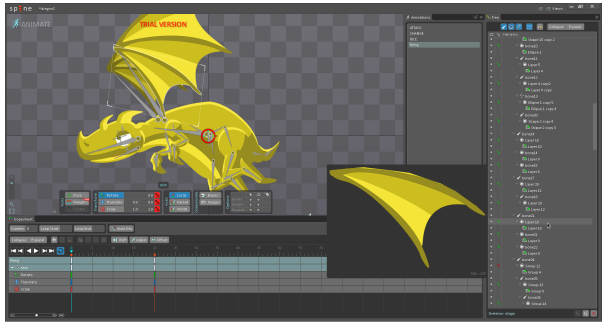
Outline

Outline

Outline

Castle Game Engine

- Game engine: combine 3D and 2D assets together.
- Scene graph is X3D.
- Open-source, modern Object Pascal language.



Rendering

Fast and modern:

- Shading: Gouraud or Phong or custom,
- Bump mapping,
- Mirrors (cubemaps and more),
- Shadows,
- Screen effects (programmable)...

High-level API for levels, creatures (ready AI), items..

This Will Be Fun

We will create a simple FPS 3D game and another 2D game using our engine.

If you have a laptop, follow us and create your own games right now too!
Download:

- Lazarus `http://www.lazarus-ide.org/`
- Castle Game Engine `http://castle-engine.sourceforge.net/engine.php`
- Example data `https://github.com/michaliskambi/cge-tutorial`

Game Data

- We support of lot of 3D and 2D formats, in particular VRML / X3D.
- Use any authoring tool you like to export to X3D.
- Actually, you can export to other formats. We support a subset of *Collada* and various other formats. But X3D is the best:)
- We have extensive support for *Spine JSON* format for 2D.

Engine API

- Access to the X3D nodes graph of your scenes.
- If you know X3D, you can immediately do a lot of stuff by processing the X3D nodes graph.
- Engine is portable. We will develop on desktop and show at the end that it works on Android too. iOS (iPhone, iPad) is possible too!
- Comfortable API for higher-level stuff. While you can just instantiate and move 3D and 2D scenes, you also have ready classes specialized for 3D level, enemies and so on.

Engine Docs

- Main site `http://castle-engine.sourceforge.net/engine.php`
- Tutorial `http://castle-engine.sourceforge.net/tutorial_intro.php`
- Reference `http://castle-engine.sourceforge.net/apidoc/html/index.html`

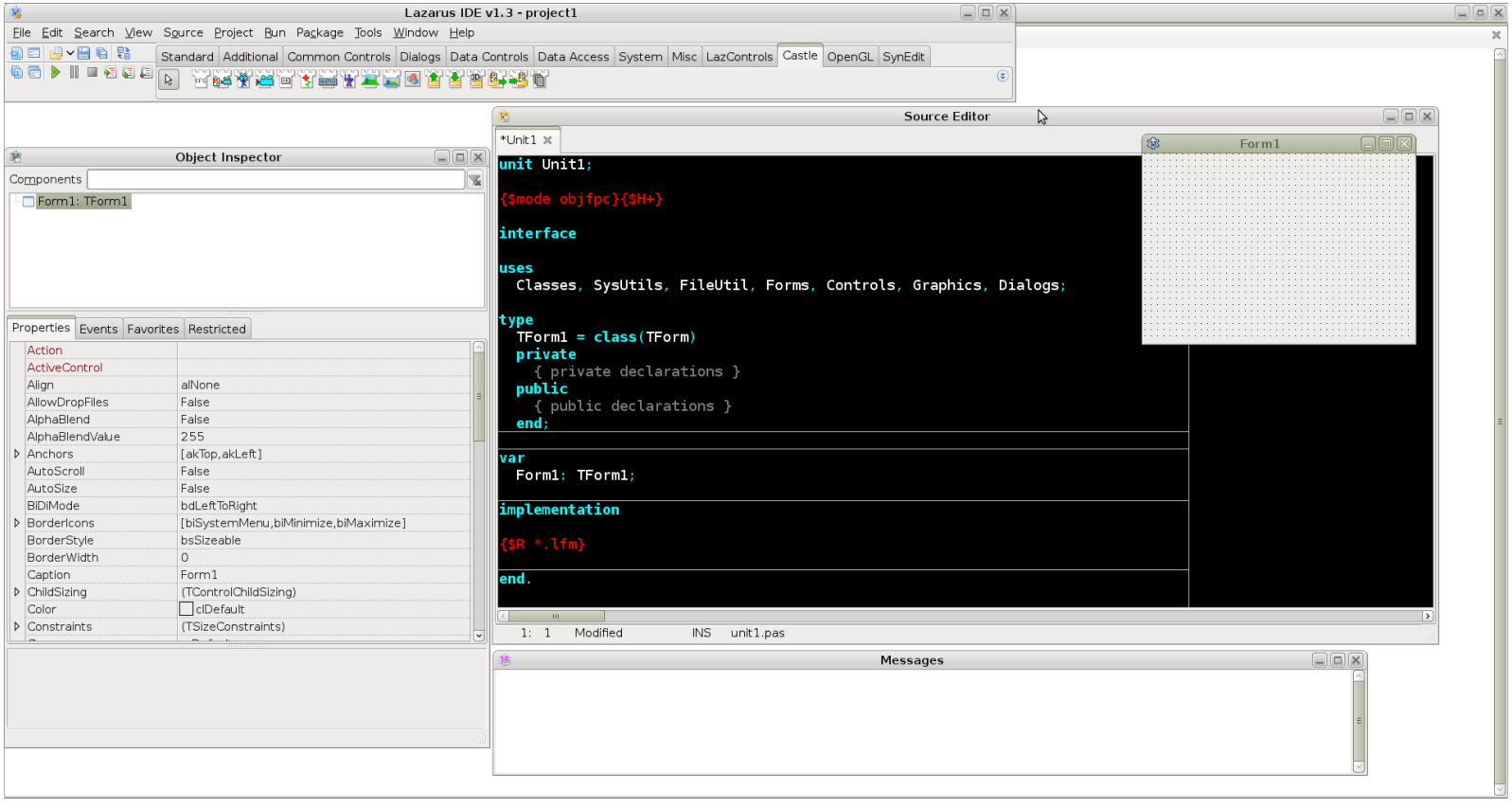
Lazarus

Lazarus `http://www.lazarus-ide.org/` is an integrated development environment, with editor, debugger and compiler (FPC) inside.

Our engine is a package for Lazarus — you install it inside Lazarus and use from there.

Outline

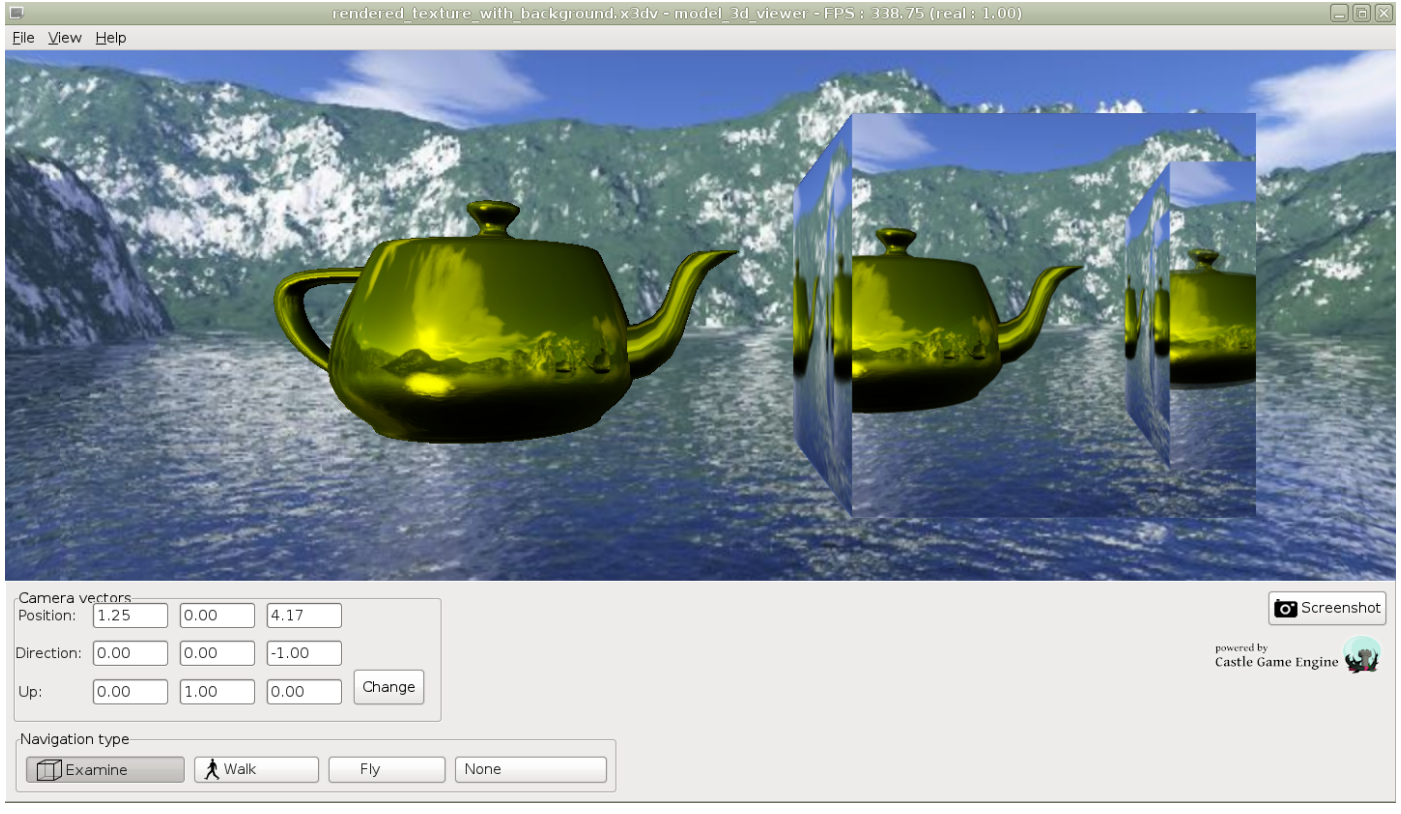
Lazarus First Peek



Engine First Peek

- Get engine from `http://castle-engine.sourceforge.net/engine.php`
- Install package `castle_components` in Lazarus.
- Compile and run `examples/lazarus/model_3d_viewer/`

Running model_3d_viewer example

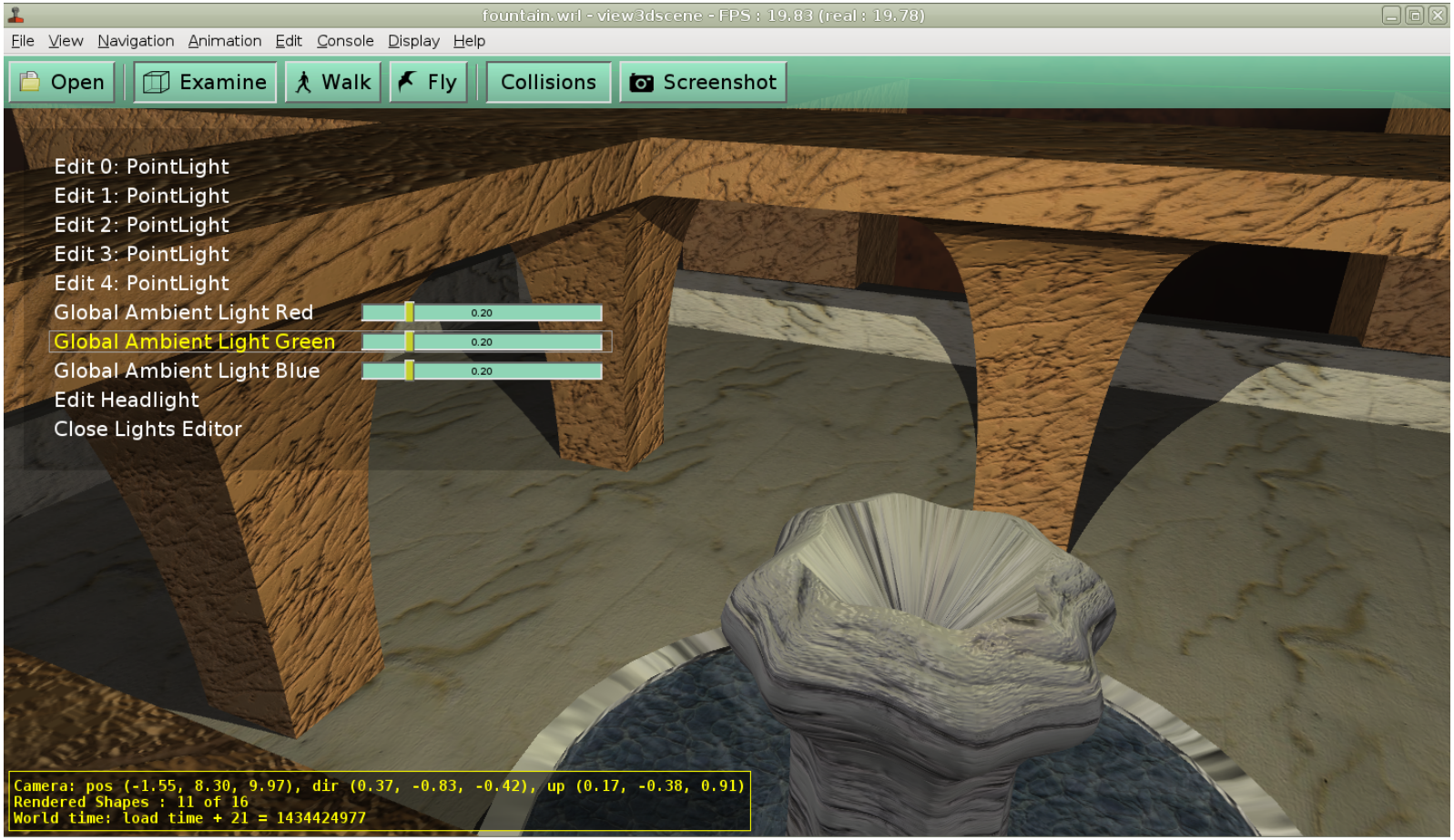


view3dscene

Full-featured 3D browser using our engine. Very useful to quickly test your 3D models before loading them to your game.

Get from `http://castle-engine.sourceforge.net/view3dscene.php`

view3dscene Lights Editor



Outline

Programming Crash Course

Very quick overview of the *Object Pascal* language.

- Probably more similar to C++ or Java than to the Pascal you learned 30 years ago:)
- Modern hybrid programming language, with everything you expect — units, classes and interfaces system, generics, rich runtime library, tools etc.
- Compiled to native, optimized code.

Example Program

```
program Example1;

procedure Foo(const Parameter: string);
begin
  if Parameter <> '' then
    WriteLn(Parameter + ' says hello!') else
    WriteLn('Parameter is an empty string.');
end;

begin
  Foo('aa');
end.
```

Example Program With Class I

```
uses SysUtils;
type
  TMyClass = class
  public
    Field: string;
    procedure MyMethod;
  end;
procedure TMyClass.MyMethod;
begin
  if Field <> '' then
    WriteIn(Field + ' says hello!') else
    WriteIn('Field is an empty string.');
end;
```

Example Program With Class II

```
var
  My: TMyClass;
begin
  My := TMyClass.Create;
  try
    My.Field := 'blah';
    My.MyMethod; { or My.MyMethod(); }
  finally FreeAndNil(My) end;
end.
```

Example Unit

```
{ exampleunit.pas }
unit ExampleUnit;

interface

procedure Foo(const Parameter: string);

implementation

procedure Foo(const Parameter: string);
begin
  if Parameter <> '' then
    WriteIn(Parameter + ' says hello!') else
    WriteIn('Parameter is an empty string.');
end;
```

Example Program Using Unit

```
{ example3.lpr }
program Example3;

uses ExampleUnit;

begin
  Foo('aa'); { or ExampleUnit.Foo('aa'); }
end.
```

Outline

Outline

Get Our Samples

Download from GitHub:
`https://github.com/michaliskambi/cge-tutorial`

If you're not sure how, just get the ZIP file and extract it:
`https://github.com/michaliskambi/cge-tutorial/archive/master.zip`

Our First Program

Let's try out Lazarus.

- Create new project using *"File -> New"*, choose "Application".
- Drop button a form.
- Add OnClick handler with this implementation:

```
ShowMessage('Hooray!');
```

Data

- Get some 3D model.
- You can use our example data from `https://github.com/michaliskambi/cge-tutorial/archive/master.zip`. Copy the `data` from `first_3d_application` directory.
- Test your scene 3D by opening it using view3dscene — `http://castle-engine.sourceforge.net/view3dscene.php`

Code

Pick `TCastleControl` from the component palette (tab *Castle*) and drop it on a regular Lazarus form.

```
uses CastleFilesUtils, CastleScene;

{ handle form OnCreate event }
procedure TForm1.FormCreate(Sender: TObject);
var
  Scene: TCastleScene;
begin
  Scene := TCastleScene.Create(Application);
  Scene.Load(ApplicationData('medkit.x3d'));

  CastleControl1.SceneManager.Items.Add(Scene);
  CastleControl1.SceneManager.MainScene:=Scene;
end;
```

Outline

Level

Multiple 3D objects can be loaded like on the last slide.

They can be composed together to form a typical game world, with level, creatures, and so on.

Specifically for **game levels** the engine includes also a special utilities. We will use it now.

Level: 3D Data

Create a 3D model for your level.

- From our example data `https://github.com/michaliskambi/cge-tutorial/archive/master.zip` copy the `data` from `fps_game` subdirectory.
- Test your level by opening it in **view3dscene**.
- Note that we use X3D Inline mechanism there: `bridge.x3d` is exported from Blender, `bridge_final.x3dv` is written by hand and inlines bridge.x3d.

Level: XML Data

Create XML file describing your level.

```
<?xml version="1.0"?>
<level
   name="bridge"
   type="Level"
   scene="bridge_final.x3dv"
   title="Bridge"
   placeholders="blender"
/>
```

Level: Code

Drop `TCastleControl` on Lazarus form.

```
uses CastleLevels;

{ handle CastleControl1 event OnOpen }
procedure TForm1.CastleControl1Open(
  Sender: TObject);
begin
  Levels.LoadFromFiles;
  CastleControl1.SceneManager.LoadLevel(
    'bridge');
end;
```

Items on Level: XML

To create a pickable item that is visible on 3D level, create resource.xml describing the item.

```
<?xml version="1.0"?>
<resource
   name="MedKit"
   type="Item"
   caption="Medic Kit"
   image="inventory_image.png">
   <model>
      <base url="medkit.x3d"/>
   </model>
</resource>
```

Items on Level: Code

Let the engine see `resource.xml` file describing the item named `MedKit`.

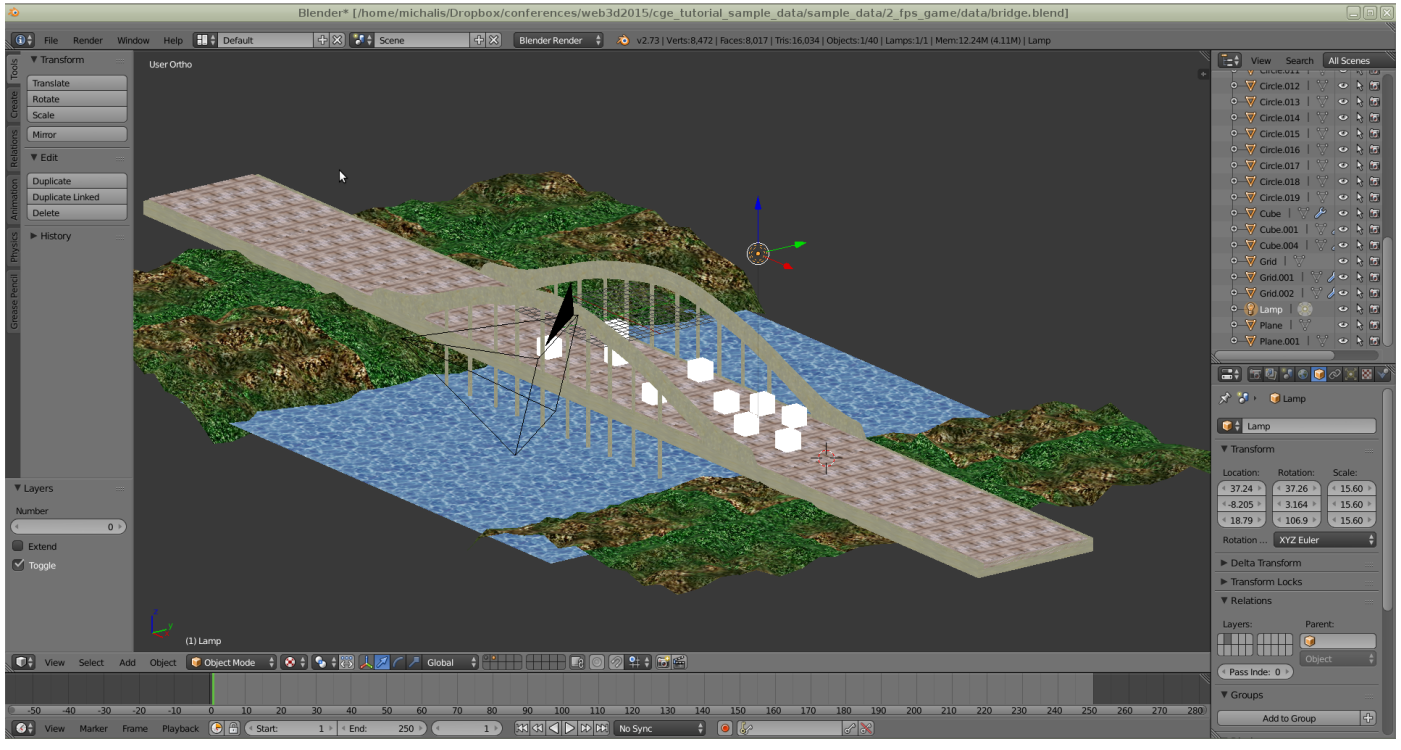
```
{ add to the implementation uses clause }
uses ..., CastleResources;

{ add at the beginning of CastleControl1Open }
Resources.LoadFromFiles;
```

Items on Level: Placeholders

You can now place in Blender special objects named `CasMedKit`

- On level load, they will be recognized and replaced with medkit 3D object.
- Item can be picked, by default will be just placed in player inventory.
- You can use placeholders for your own stuff as well.
- Placeholder detection can be adjusted for given needs or 3D authoring software.

Placeholders in Blender



Player

There is no player yet (only a default camera, that does not have inventory or other game features).

```
{ add to implementation uses clause }
uses ..., ,CastlePlayer;

{ declare in class private section }
Player: TPlayer;

{ add at the beginning of CastleControl1Open }
Player := TPlayer.Create(
  CastleControl1.SceneManager);
CastleControl1.SceneManager.Items.Add(Player);
CastleControl1.SceneManager.Player := Player;
```

HUD I

Items are pickable now, they are added to player's inventory. But the inventory is not visualized in any way. Add a simple 2D HUD:

```
{ add to implementation uses clause }
uses ..., , CastleUIControls;

{ add to implementation }
type
  TGame2DControls = class(TUIControl)
  public
    procedure Render; override;
  end;

procedure TGame2DControls.Render;
```

HUD II

```
var
  Player: TPlayer;
  I, J, X: Integer;
begin
  Player := Form1.Player;
  X := 0;
  for I:=0 to Player.Inventory.Count-1 do
    for J:=0 to Player.Inventory[I].Quantity-1 do
    begin
      Player.Inventory[I].Resource.GLImage.
        Draw(X, 0);
      X += 100;
    end;
end;
```

HUD III

```
{ add to CastleControl1Open }
var
  Game2DControls: TGame2DControls;
begin
  ...
  Game2DControls := TGame2DControls.Create(
    Application);
  CastleControl1.Controls.InsertFront(
    Game2DControls);
end;
```

Creatures

Adding a creature with a predefined AI does not require any code at all.

- We already load resources (items and creatures) by `Resources.LoadFromFiles`.
  This detects `knight_creature/resource.xml` file inside as resource named `Knight` using a standard AI called `WalkAttack`.
- We already load the level using `LoadLevel` facility.
  This allows to place initial creatures on level by placing `CasResKnight` from **Blender**.

Spawn Creatures I

Add a button on a form to spawn creatures. Best to use `TSpeedButton`, to not capture focus.

```
{ add to implementation uses clause }
uses ..., CastleVectors, CastleCreatures;

{ handle button OnClick event }
procedure TForm1.Button1Click(Sender: TObject);
var
  P: TVector3Single;
  Direction: TVector3Single;
  CreatureResource: TCreatureResource;
begin
  P := Player.Position + Player.Direction * 10;
  Direction := Player.Direction;
```

Spawn Creatures II

```
  CreatureResource :=
    Resources.FindName('Knight')
    as TCreatureResource;
  CreatureResource.CreateCreature(
    CastleControl1.SceneManager.Items,
    P, Direction);
end;
```

Kill Creatures I

Add a button to kill creature.

```
{ add to implementation uses clause }
uses ..., Castle3D;

{ handle kill button OnClick event }
procedure TForm1.SpeedButton2Click(
  Sender: TObject);
var
  I: Integer;
  Hit: TRayCollision;
begin
  Hit := Player.Ray(Player.Middle,
    Player.Direction);
```

Kill Creatures II

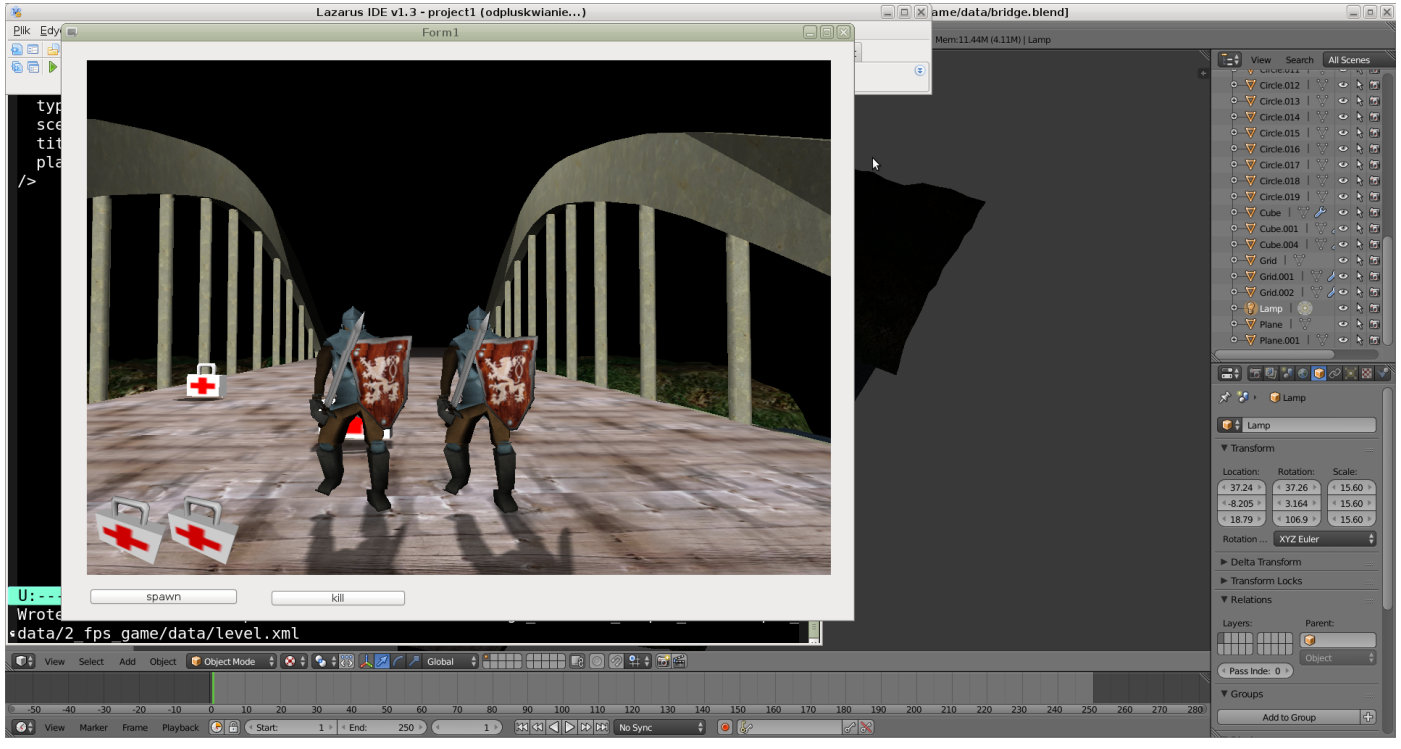
```
  if Hit <> nil then
  begin
    for I := 0 to Hit.Count - 1 do
      if Hit[I].Item is T3DAlive then
      begin
        (Hit[I].Item as T3DAlive).Hurt(
          100, Player.Direction, 1, Player);
        Break;
      end;
    FreeAndNil(Hit);
  end;
end;
```

Shadows

Since all our 3D data is in X3D, we can do a lot of fun stuff with X3D coding. Use Castle Game Engine X3D extensions to get shadows, screen effects, shader effects, mirrors, and more.

- For a simplest demo, enable shadows by „shadows TRUE” on a light source.
- In our sample data from `https://github.com/michaliskambi/cge-tutorial/archive/master.zip` just uncomment relevant lines at the end of `bridge_final.x3dv`.

Our Game With Shadows



Find X3D Node I

We can trivially access X3D node and change it's property from Object Pascal. Notice the `DEF Sun` in X3D file.

```
{ add to interface uses clause }
uses ..., X3DNodes;

{ declare in class private section }
Root: TX3DRootNode;
Sun: TSpotLightNode;

{ add to CastleControl1Open }
Root := CastleControl1.SceneManager.
  MainScene.RootNode;
Sun := Root.FindNodeByName(TSpotLightNode,
  'Sun', true) as TSpotLightNode;
```

Modify X3D Field

We have a reference to our `Sun` X3D node, we can modify it's fields.

```
{ handle CastleControl1 event OnUpdate }
procedure TForm1.CastleControl1Update(
  Sender: TObject);
var
  V: TVector3Single;
begin
  V := Sun.Location;
  V[2] := Sin(CastleControl1.SceneManager.
    MainScene.Time.Seconds) * 10;
  Sun.Location := V;
end;
```

Outline

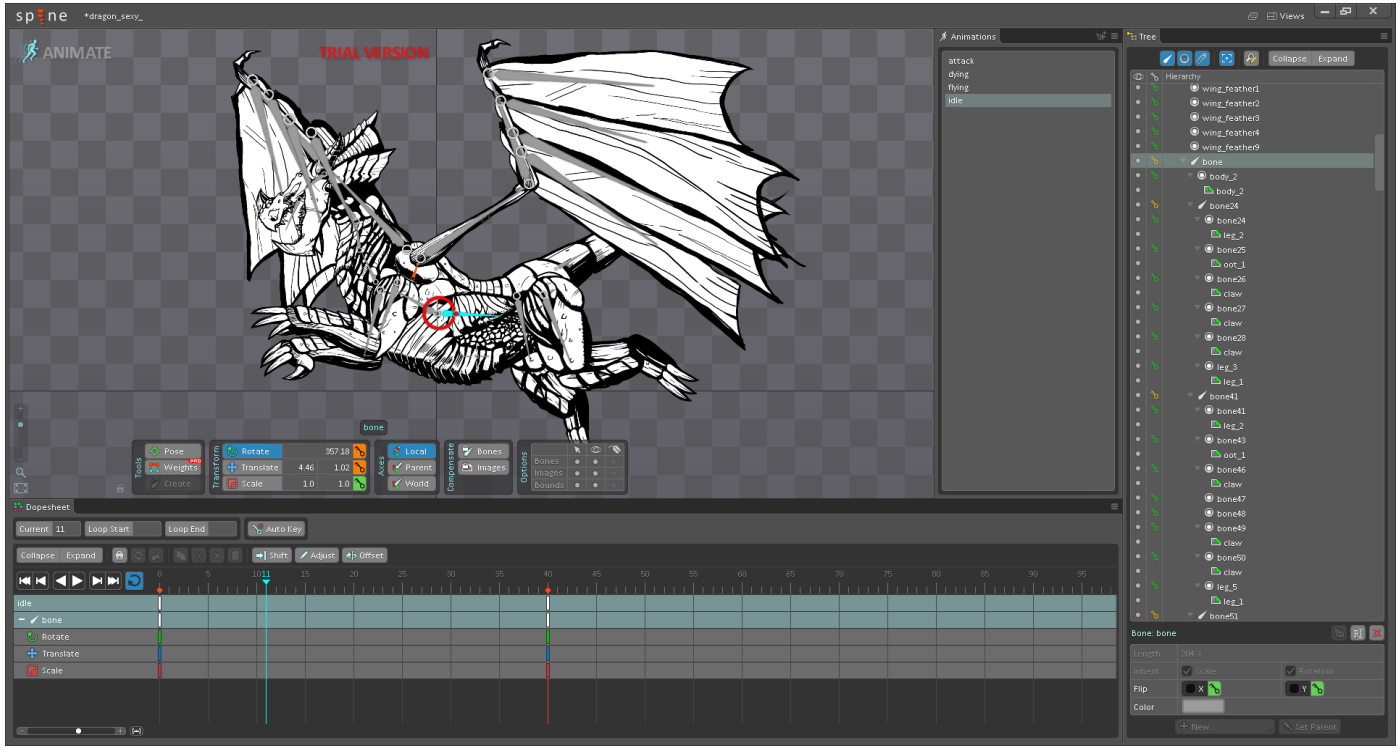
Data

Example data on `https://github.com/michaliskambi/cge-tutorial/archive/master.zip`, copy `data` from `2d_game` subdirectory.

Open with view3dcene `background.x3dv` and `dragon/dragon.json`. Look at `background.x3dv` with a text editor, it's really trivial X3D file.

Dragon was modelled in Spine, and exported to JSON. Our engine can read it (it's converted to X3D graph of course).

Spine Peek


spine
*dragon_sexy_
ANIMATE
TRIAL VERSION
Animations
attack
dying
flying
idle
Tree
Collapse
Expand
Hierarchy
wing_feather1
wing_feather2
wing_feather3
wing_feather4
wing_feather9
bone
body_2
bone24
leg_2
bone25
oot_1
bone26
claw
bone27
bone28
leg_3
leg_1
bone41
bone43
bone46
bone47
bone48
bone49
bone50
leg_5
bone51
Bone: bone
Dopesheet
Current
Loop Start
Loop End
Auto Key
Shift
Adjust
Offset
Rotate
Translate
Scale
Pose
Weights
Create
Local
Parent
World
Bones
Images
Bounds

Code I

Drop `TCastle2DControl` on a form.

```
{ add to interface uses clause }
uses ..., CastleFilesUtils, Castle2DSceneManager;

{ declare as form private field }
Background: T2DScene;

{ handle form OnCreate }
procedure TForm1.FormCreate(Sender: TObject);
var
  SM: T2DSceneManager;
begin
  Background := T2DScene.Create(Application);
  Background.Load(
```

Code II

```
    ApplicationData('background.x3dv'));

  SM := Castle2DControl1.SceneManager;
  SM.Items.Add(Background);
  SM.MainScene := Background;
  SM.ProjectionAutoSize := false;
  SM.ProjectionHeight := 721;
end;
```

Dragon

Load a dragon and add it to the world. Start `flying` animation immediately.

```
{ declare as form private field }
Dragon: T2DScene;

{ add to FormCreate }
Dragon := T2DScene.Create(Application);
Dragon.Load(
  ApplicationData('dragon/dragon.json'));
Dragon.ProcessEvents := true;
Dragon.PlayAnimation('flying', paForceLooping);
Castle2DControl1.SceneManager.Items.Add(Dragon);
```

Sidenote: Screen Size

We don't worry about form size or screen size, because everything adjusts to the form size automatically.

We had to set the projection size, which says what portion of game world is visible through the viewport. We don't worry how many pixels has the viewport.

Scaling The Dragon I

It would be more useful to have some control over dragon position and scale. We could wrap it in X3D `Transform` node. Or use `T3DTransform` that does something similar, but is more comfortable in this case.

```
{  add to interface uses clause }
uses ..., Castle3D, CastleVectors;

{ declare as private field in TForm1 }
DragonTransform: T3DTransform;

{ add to FormCreate }
DragonTransform :=
  T3DTransform.Create(Application);
DragonTransform.Scale :=
```

Scaling The Dragon II

```
  Vector3Single(0.2, 0.2, 0.2);
Castle2DControl1.SceneManager.Items.Add(
  DragonTransform);
{ ... here create Dragon, like previously,
  but add it to DragonTransform: }
DragonTransform.Add(Dragon);
```

Capturing Mouse I

Let's capture mouse clicks to know where user wants to move the dragon.

```
{ declare as private field in TForm1 }
FlyingTarget: TVector2Single;

{ in FormCreate add }
Background.Spatial := [ssRendering,
  ssDynamicCollisions];
Background.ProcessEvents := true;

{ handle assign Castle2DControl1 event OnPress}
procedure TForm1.Castle2DControl1Press(
  Sender: TObject;
  const Event: TInputPressRelease);
```

Capturing Mouse II

```
begin
  if Event.IsMouseButton(mbLeft) and
     (Background.
       PointingDeviceOverItem <> nil) then
    FlyingTarget := Vector2Single(
      Background.PointingDeviceOverPoint[0],
      Background.PointingDeviceOverPoint[1]);
end;
```

Note: this was one point where we have to remember that viewport presents the world, and world coordinates (in Background.PointingDeviceOverPoint) are not simple mouse position (in Event.Position).

Sidenote: How To Log

We coded the last part *blind*. Would be nice to see what is going on, whether `FlyingTarget` gets any sensible values.

Show message in Lazarus:

```
ShowMessage('my string');
ShowMessage('my vector is ' +
  VectorToNiceStr(MyVector));
ShowMessage(Format('my int %d, my float %f',
  [MyInt, MyFloat]));
```

Moving The Dragon Immediately

Using `DragonTransform` we can easily move the whole dragon 3D object.

```
procedure TForm1.Castle2DControl1Update(
  Sender: TObject);
var
  T: TVector3Single;
begin
  T := DragonTransform.Translation;
  T[0] := FlyingTarget[0];
  T[1] := FlyingTarget[1];
  DragonTransform.Translation := T;
end;
```

Moving The Dragon With Speed I

Many ways to do it.

```
function MoveTo(const Start, Target: Single):
  Single;
var
  MoveDist: Single;
begin
  MoveDist := 500 *
    Castle2DControl1.Fps.UpdateSecondsPassed;
  if Start < Target then
    Result := Min(Target, Start + MoveDist) else
    Result := Max(Target, Start - MoveDist);
end;
```

Moving The Dragon With Speed II

```
var
  T: TVector3Single;
begin
  T := DragonTransform.Translation;
  T[0] := MoveTo(T[0], FlyingTarget[0]);
  T[1] := MoveTo(T[1], FlyingTarget[1]);
  DragonTransform.Translation := T;
  { see example code for additional
    trick to horizontally mirror dragon }
end;
```

Outline

Outline

What We Know Now

This was just a start of a 3D, and then a short start of a 2D game, but I'm sure you get the idea how to continue.
You have all the necessary tools now.

- You can manage levels, player, items, inventory, creatures on a 3D world.
- You can directly create and destroy and move whole 2D and 3D objects (TCastleScene, T2DScene).
- You can modify the scene's X3D graph.

Outline

Our Game On Android

Android

- Need to use `TCastleWindow` engine component, not `TCastleControl`. Like `TCastleControl` but engine fills 100
- The engine supports custom 3D viewports and 2D controls. So your game area is still fully configurable. You just have to draw 100
- It is easy to create game code that compiles and works on both desktop (Windows, Linux, Mac OS X...) and Android.
- See example code of `2d_game_android_and_desktop`.

Build tool

The engine includes a build tool, that compiles a production-ready Android apk (something you can install or upload to Google Play). You only provide a Pascal source code, and CastleEngineManifest.xml declaring basic application parameters.

See example `CastleEngineManifest.xml`.

Integration with Android Libraries

The engine code can easily interact with Java on Android. Some Android APIs with which we already successfully integrated:

- Google Play Game Services,
- In-app payments,
- Ads,
- Google Analytics,
- Game Analytics.

For some cool games done using Castle Game Engine, published on Google Play, visit `https://play.google.com/store/apps/developer?id=Michalis+Kamburelis`

Outline

Visit us:

http://castle-engine.sourceforge.net/

Thank you for your attention!

Questions?