OMB EXPLODED!

Adding Zig support to the Defold game engine

Today

- What is Defold?
- Why Zig?
- Case Study for Defold
- Conclusions
- Q & A

Who are we?

Björn Ritzl
@bjornritzl
@britzl@mastodon.gamedev.place
bjorn@defold.se

Mathias Westerdahl
@mwesterdahl76
@jcash@mastodon.gamedev.place
mathias@defold.se

Defold Editor 2.0 - Hamster
Assets
boss.spinemodel
Outline
Console
Curve Editor
Build Errors
Search Results
Welcome to Defold!
Changed Files
DEFOLD
Defold Editor 2.0 - GeometryWars
player.go
tail.particlefx
Properties

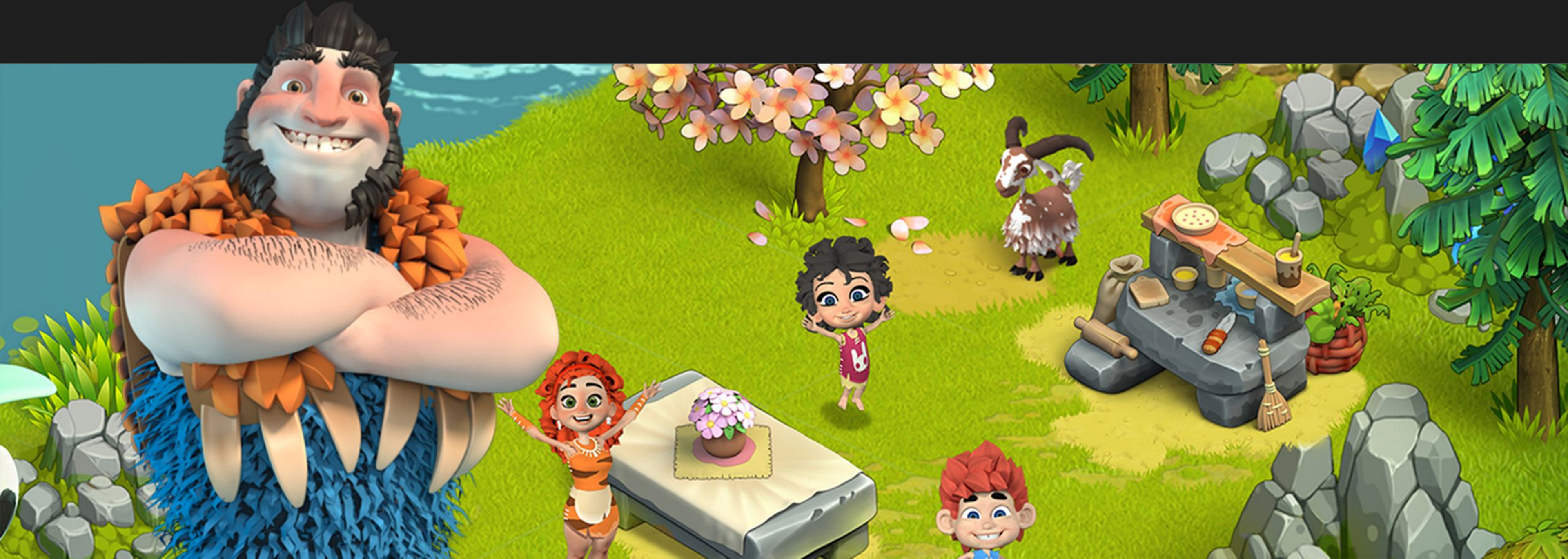
History

- Created in 2009
  - By Christian Murray and Ragnar Svensson
- Bought by King in 2014
  - Released a few games
- Made available for free in 2016
- Made source available in 2020
- Source + Trademarks transferred to Defold Foundation in 2020

FAMILY ISLAND

MELSOFT GAMES

1M DAU - 4M MAU



Why Zig?

- Seems interesting / promising
- We use a C-like-C++ api
- Focus on small runtimes
- Defold extension system
  - Cloud build server (https://github.com/defold/extender)
  - Currently 8 platforms (desktop, mobile, html5 & console)
    - Should work well to add Zig
- A bit of fun!

What do we have? - *the cloud builder*

- Extension System
  - User supplied custom code
    - C/C++, Java, Objective C
  - Upload user code to cloud server
  - Server builds static libraries
  - Links engine
  - Sends custom engine back to user

```
# Zig
zigSourceRe:    '(?i).*(\.zig)'
zigCompileCmd:  '{{env.ZIG_PATH}}/zig build-obj -target {{zig-target}} {{#ext.includes}}-I{{{.}}} {{/ext.includes}} {{#include
```

What do we have? - *the Defold SDK*

- Defold SDK
  - Allows user to add…
    - custom component & resource types
    - custom Lua modules
  - Mostly C-like-C++
    - Few container templates (vector + hashtable)
    - Namespaces
    - RAII (mutex locks, profile scopes)
  - Currently 101 headers

Zig support - What do we need?

- Zig lives in C land
  - We need a C api
- Backwards compatibility
  - 100+ C++ plugins in the wild
- Minimal Maintenance
  - C api needs to match C++ api
  - Easy to update our api

C API - some options (for us)

- Manually port our headers to C
  - We can have more control over the code
  - Cons: 100+ headers!
- Code generation
  - Keeps api's in sync
  - Cons: Takes time to R&D and configure
- Experimental C++ wrapper (c2z)
  - Cons:
    - Very early stages
    - Doesn't give us a C api

Porting our API to C

- Constructors/Deconstructors
  - To initialize values, but also for RAII
- Templates
  - Used in API calls and structs (POD types)
  - Allows users to use same containers
- Namespaces
- Enums

Getting down to business

- Disclaimer:
  - Quick and dirty R&D, only scratching the surface of Zig
  - Goal: Get a happy path working!
  - We’re not looking to replace our engine code
  - We want to allow users writing plugins using Zig

Getting down to business

- First approach: Manual C approach
  - 2 api's in the same header
- One api calls the other
  - C api calls C++ api to keep 100% sync

Getting down to business - Enums

- C

```
typedef enum dmExtensionEventID {
   DM_EXTENSION_EVENT_ID_ACTIVATEAPP,
   DM_EXTENSION_EVENT_ID_DEACTIVATEAPP,
   DM_EXTENSION_EVENT_ID_ICONIFYAPP,
   DM_EXTENSION_EVENT_ID_DEICONIFYAPP,
} dmExtensionEventID;
```

- C++

```
namespace dmExtension {
  enum EventID {
       EVENT_ID_ACTIVATEAPP     = DM_EXTENSION_EVENT_ID_ACTIVATEAPP,
       EVENT_ID_DEACTIVATEAPP   = DM_EXTENSION_EVENT_ID_DEACTIVATEAPP,
       EVENT_ID_ICONIFYAPP      = DM_EXTENSION_EVENT_ID_ICONIFYAPP,
       EVENT_ID_DEICONIFYAPP    = DM_EXTENSION_EVENT_ID_DEICONIFYAPP
   };
}
```

Getting down to business - Typedefs

- C

```
typedef struct dmConfigFileConfig* dmConfigFileHConfig;
```

- C++

```
namespace dmConfigFile {
    typedef dmConfigFileHConfig HConfig;
}
```

Getting down to business - Constructors

- C

```
void dmExtensionParams_Init (dmExtensionParams* params) {
    memset(params, 0, sizeof(*params));
}
```

- C++

```
dmExtensionParams::dmExtensionParams () {
    memset(this, 0, sizeof(*this));
}
```

Getting down to business - Destructors

- Used for RAII
  - Mutexes
  - Profile scopes
- We already have the C-like functions for this
  - E.g. MutexLock() / MutexUnlock
- Zig
  - Either we generate Zig specific helpers
  - Or the developer uses our C functions as-is

Getting down to business - vectors

- Mostly used in the sdk for resource type structs
    - We could use C arrays
    - Can we create an api that maps on top of the ABI?
    - Protobuf is C++
    - Hiding behind opaque pointer seems easiest

- C++

```
dmArray<SomeType> data;
```

- C

```
SomeType* GetData(void* resource, uint32_t* data_count);
```

Getting down to business - hash table

- Same problems as vectors
  - Can also be solved with opaque struct + data accessors

- C++

```
dmHashTable<uint64_t, SomeType*> sub_contexts;
```

- C

```
void* GetSubContext(Context* ctx, const char* name);
```

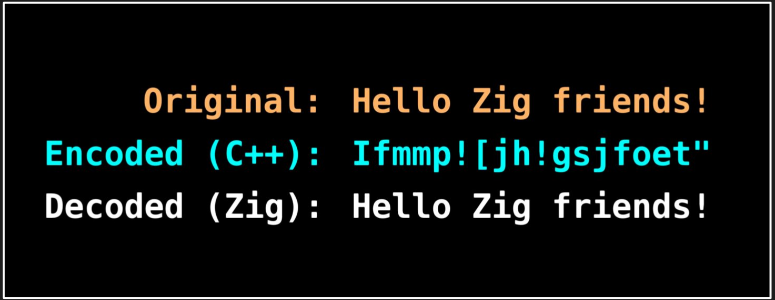
Demo

Steps:

- Clone repo: https://github.com/defold/example-zig
- Open in Defold (experimental version)
- Press “Build and Run”

Description

- The example encodes+decodes a string in Lua
  - Encoder is written in C++, and adds 1 to each character: “Ac” -> “Bd”
  - Decoder is written in Zig, and subtracts 1 from each character: “Bd” -> “Ac”



```
     Original:  Hello Zig friends!
Encoded (C++):  Ifmmp![jh!gsjfoet"
Decoded (Zig):  Hello Zig friends!
```

Conclusion

- Learning curve
  - Memory allocation
  - Pointers / Strings
- The Zig experience is a bit rough around the edges
  - Build errors (0.10)
    - Obscure messages (hard to understand what to do)
    - If C header contained errors, *cInclude* would not report as-is
  - Documentation
    - Took a long time to find out about “build-obj”, and no real examples
    - No documentation about linksection
    - Special thanks to the Zig Discord for helping out!
    - “Hidden” documentation (*reddit*)

Conclusions

- The Zig experience is a bit rough around the edges
- Porting would likely be split between manual work and code generation
  - Not straightforward!
- Main benefits will be for new plugins
  - Core engine code will remain C++
  - A lot of work needed to fully support our SDK
- But overall success!

Future

- Port more SDK headers to C
  - E.g. add new game component/resource types
- Integrate Zig testing into the build step
- Use a Zig package manager

What I’d like to see

- Migration guides
  - How to convert 3rd party libraries and integrate into existing code bases
  - How to incorporate zig libraries into C/C++ code
  - Memory / allocators / containers best practices to/from Zig
- Documentation and examples on official web page
  - Less reliance on 3rd party sites
- Crazy idea: Some kind of *cppImport(“api.hxx”)*
  - For C-like-C++ code (e.g no templates)

Thank you!

Q & A

- Links
  - www.defold.com
  - github.com/defold/defold
- Social
  - https://twitter.com/defold
  - https://mastodon.gamedev.place/@defold