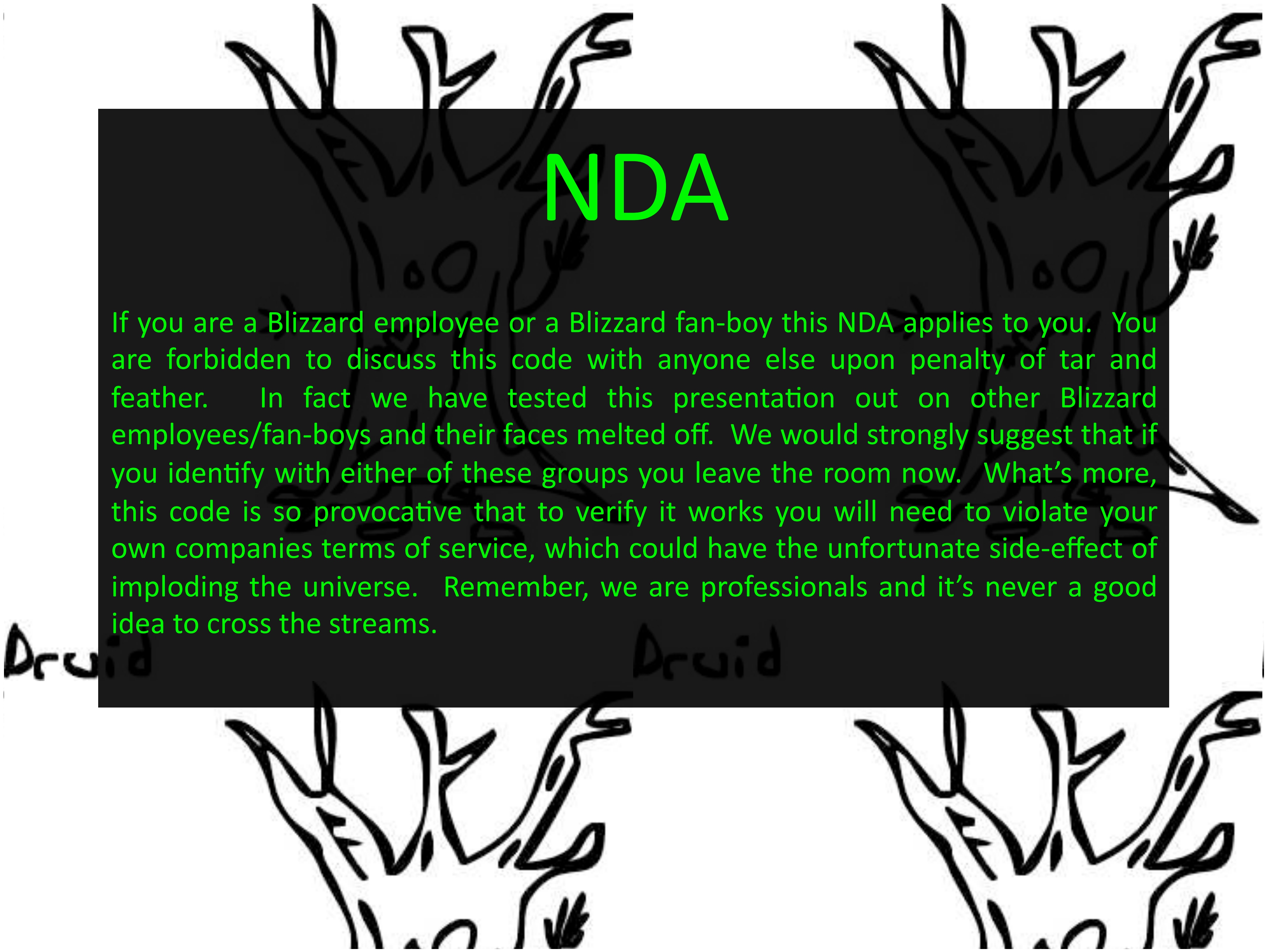
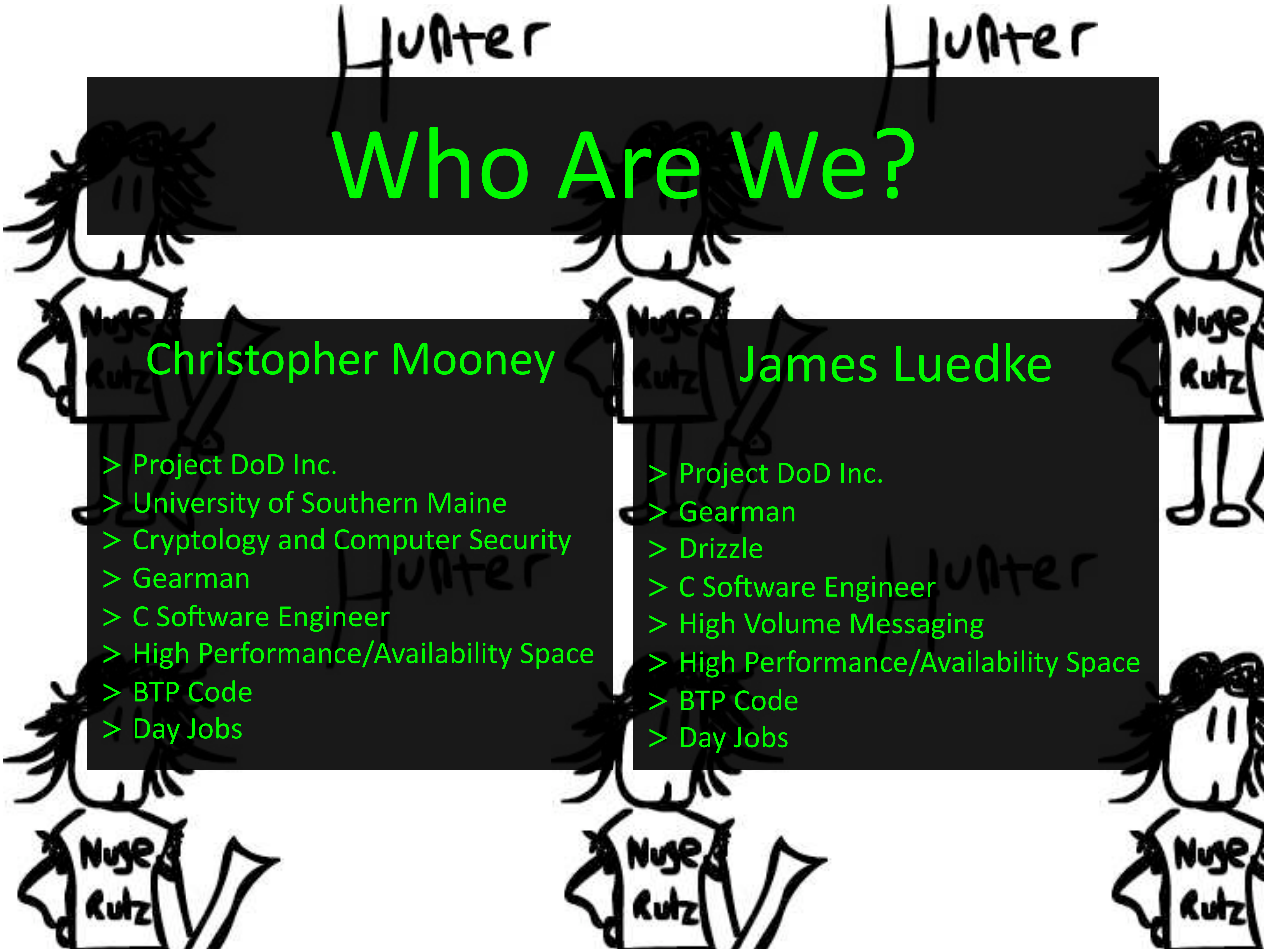
Subverting the World of Warcraft API

by
Christopher Mooney
James Luedke



NDA

If you are a Blizzard employee or a Blizzard fan-boy this NDA applies to you. You are forbidden to discuss this code with anyone else upon penalty of tar and feather. In fact we have tested this presentation out on other Blizzard employees/fan-boys and their faces melted off. We would strongly suggest that if you identify with either of these groups you leave the room now. What's more, this code is so provocative that to verify it works you will need to violate your own companies terms of service, which could have the unfortunate side-effect of imploding the universe. Remember, we are professionals and it's never a good idea to cross the streams.



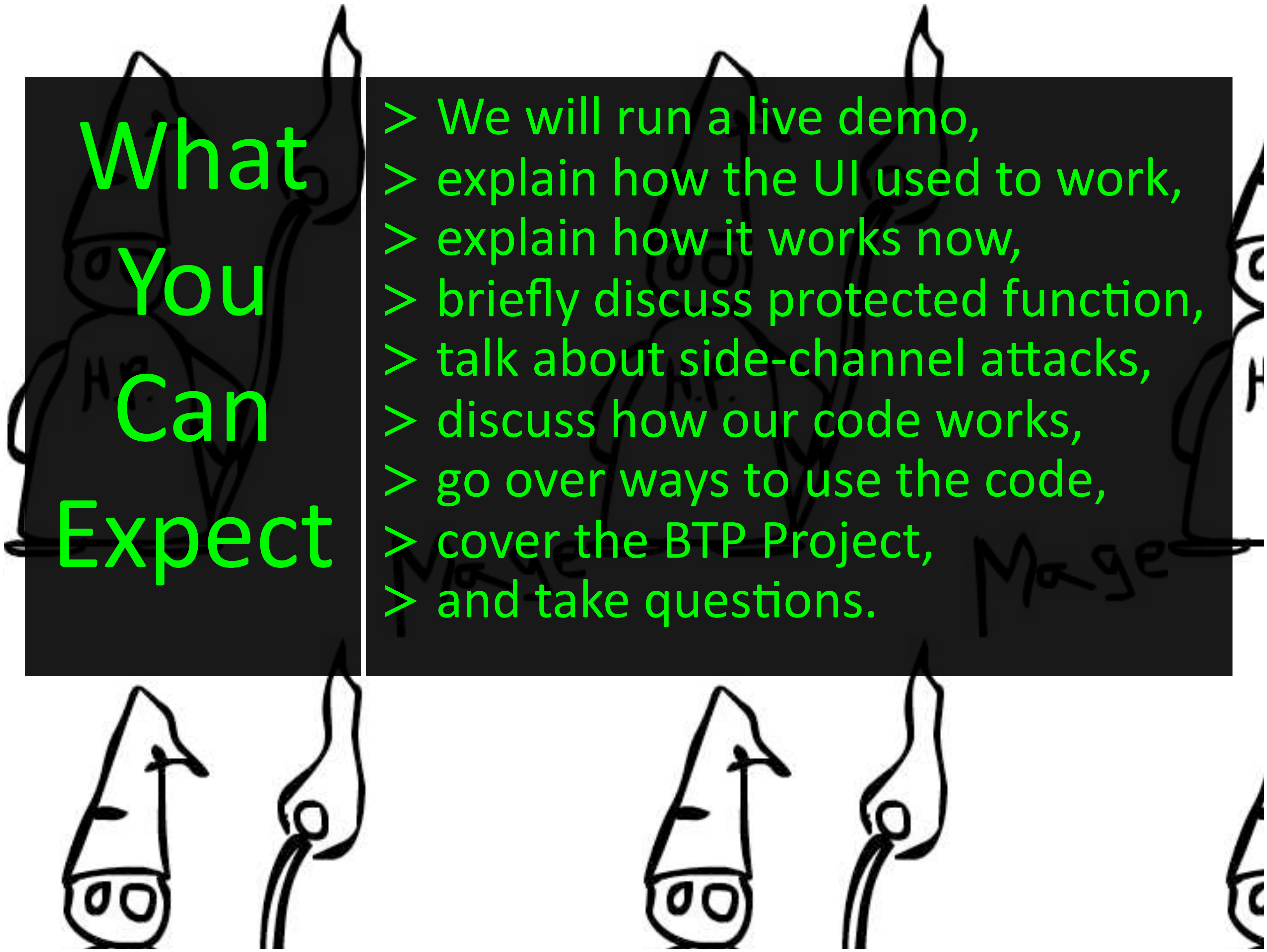
Who Are We?

Christopher Mooney

> Project DoD Inc.
> University of Southern Maine
> Cryptology and Computer Security
> Gearman
> C Software Engineer
> High Performance/Availability Space
> BTP Code
> Day Jobs

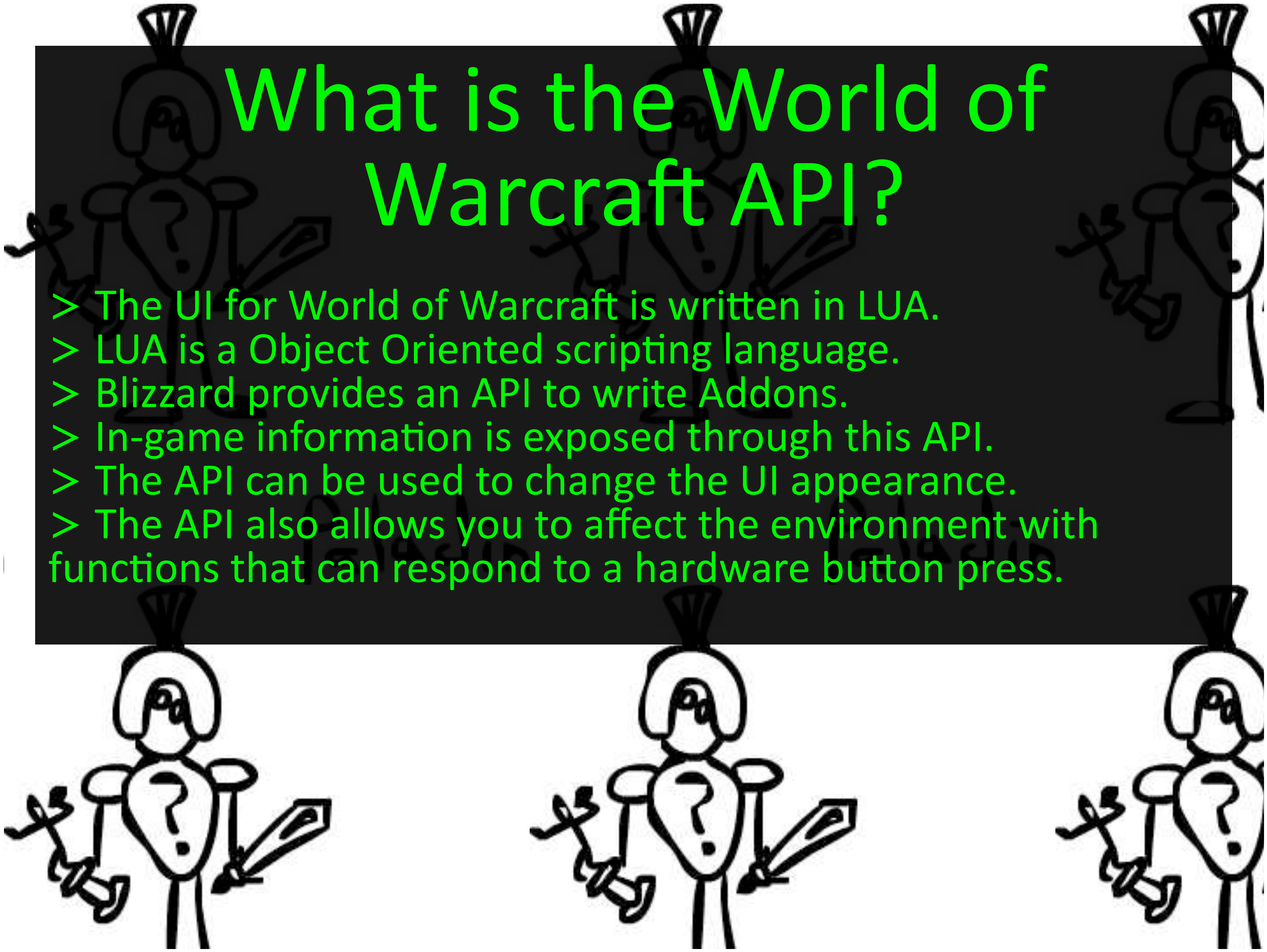
James Luedke

> Project DoD Inc.
> Gearman
> Drizzle
> C Software Engineer
> High Volume Messaging
> High Performance/Availability Space
> BTP Code
> Day Jobs



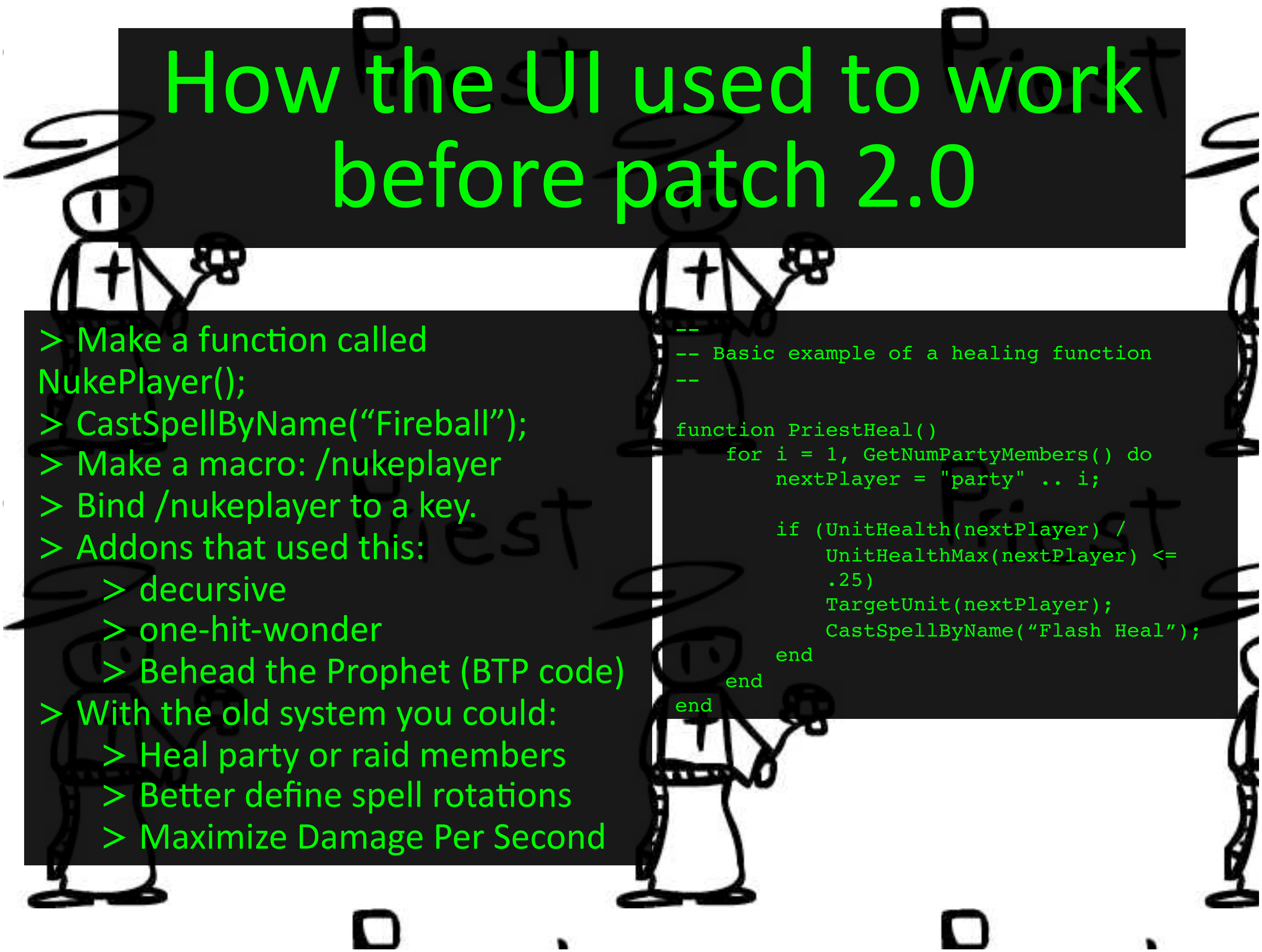
What You Can Expect

> We will run a live demo,
> explain how the UI used to work,
> explain how it works now,
> briefly discuss protected function,
> talk about side-channel attacks,
> discuss how our code works,
> go over ways to use the code,
> cover the BTP Project,
> and take questions.



What is the World of Warcraft API?

> The UI for World of Warcraft is written in LUA.
> LUA is a Object Oriented scripting language.
> Blizzard provides an API to write Addons.
> In-game information is exposed through this API.
> The API can be used to change the UI appearance.
> The API also allows you to affect the environment with functions that can respond to a hardware button press.



How the UI used to work before patch 2.0

- > Make a function called NukePlayer();
- > CastSpellByName(“Fireball”);
- > Make a macro: /nukeplayer
- > Bind /nukeplayer to a key.
- > Addons that used this:
    - > decursive
    - > one-hit-wonder
    - > Behead the Prophet (BTP code)
- > With the old system you could:
    - > Heal party or raid members
    - > Better define spell rotations
    - > Maximize Damage Per Second

```
--
-- Basic example of a healing function
--

function PriestHeal()
    for i = 1, GetNumPartyMembers() do
        nextPlayer = "party" .. i;

        if (UnitHealth(nextPlayer) /
            UnitHealthMax(nextPlayer) <=
            .25)
            TargetUnit(nextPlayer);
            CastSpellByName(“Flash Heal”);
        end
    end
end
```

How the UI works now on patch 3.x

>Removed access to a bunch of API functions.
> TargetUnit()
> CastSpellByName()
> Prevented some functions from being called in combat.
> SetBindingClick()
> Protected functions that could only be executed by Blizzard signed code were introduced.
> Community complaints fell on def ears.
> Users and coders alike had to endure a paradigm shift.
>Addons that broke because of this change:
> decursive
> one-hit-wonder
> Behead the Prophet (BTP code)
> Users could no longer:
> Heal party or raid members programmatically
> Use LUA to define spell rotations
> Decurse programmatically
> Target programmatically

```
--
-- Basic example of a healing function
--

function PriestHeal()
    for i = 1, GetNumPartyMembers() do
        nextPlayer = "party" .. i;

        if (UnitHealth(nextPlayer) /
            UnitHealthMax(nextPlayer) <=
            .25)
            TargetUnit(nextPlayer);
            CastSpellByName(“Flash Heal”);
        end
    end
end
```

Why did Blizzard Make These Changes?

> Our understanding is that they thought programmatic decision making unbalanced the game.

> There may be a larger philosophy behind why they chose to do this.

> Maybe they just hate us.

New Terms of Service Changes

> Add-ons must be free of charge.

> Add-on code must be completely visible.

> Add-ons must not negatively impact World of Warcraft realms or other players.

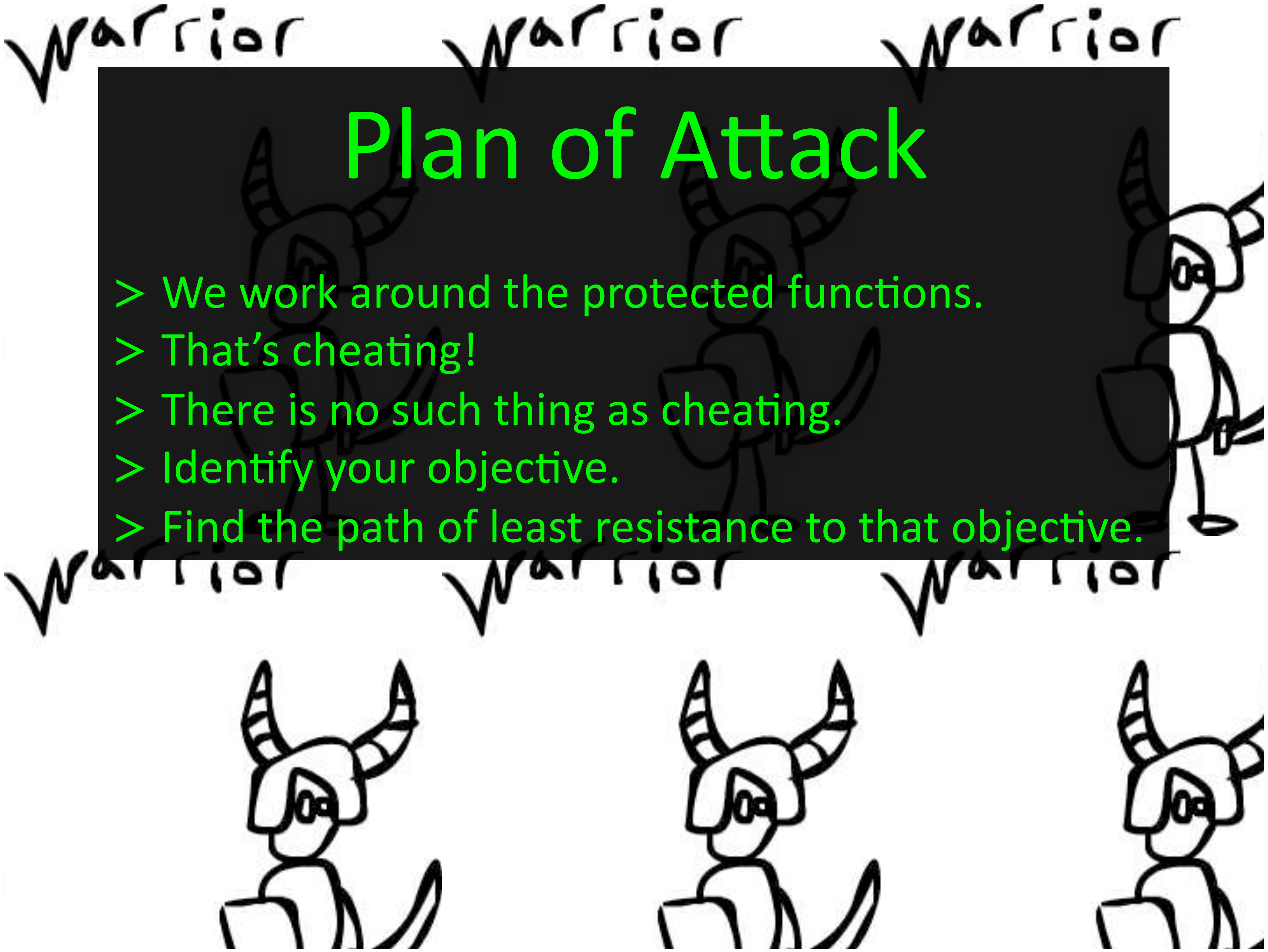
> Add-ons may not include advertisements.

> Add-ons may not solicit donations.

> Add-ons must not contain offensive or objectionable material.

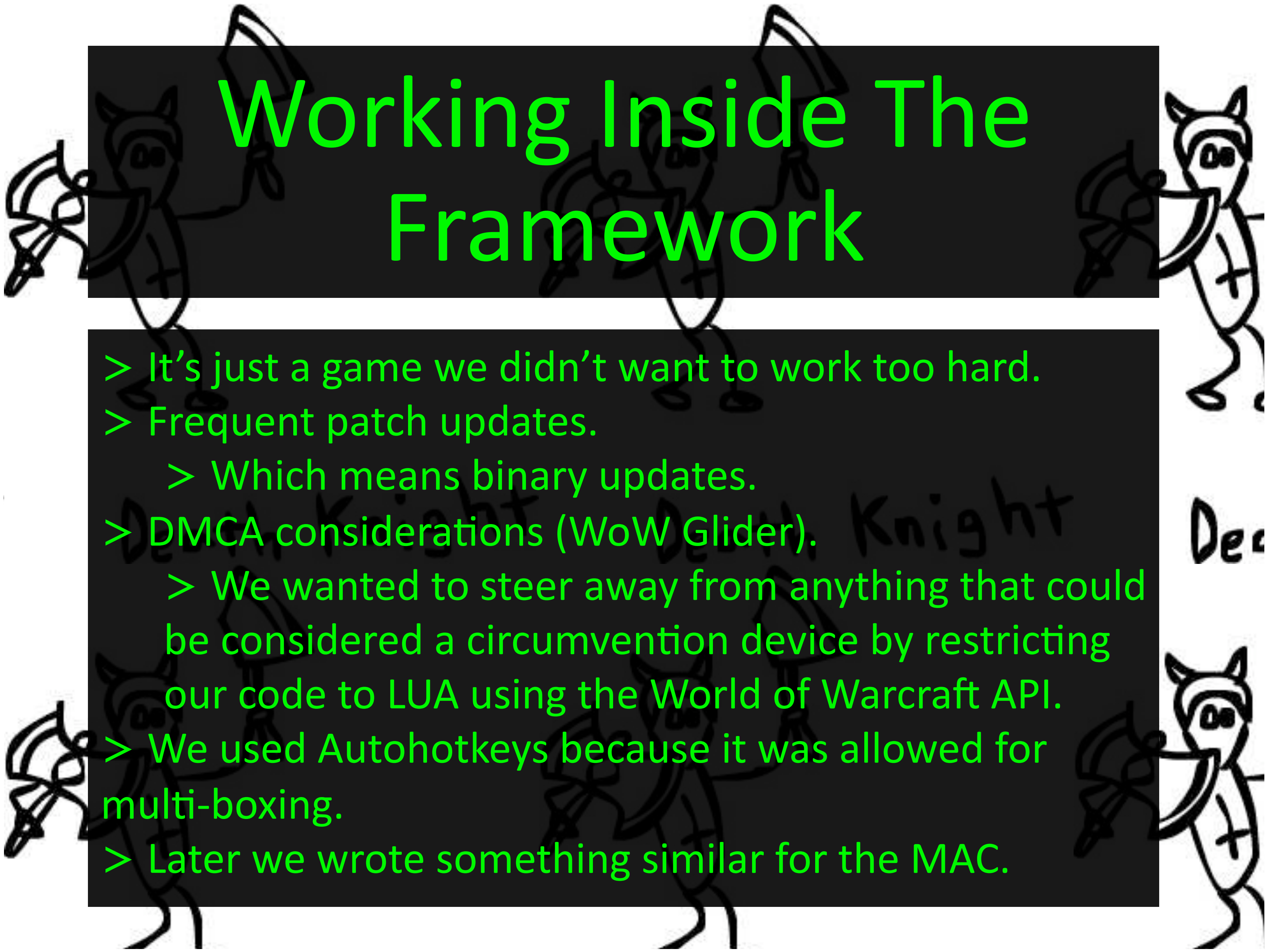
> Add-ons must abide by World of Warcraft ToU and EULA.

> Blizzard Entertainment has the right to disable add-on functionality as it sees fit.



Plan of Attack

> We work around the protected functions.
> That's cheating!
> There is no such thing as cheating.
> Identify your objective.
> Find the path of least resistance to that objective.



Working Inside The Framework

- It's just a game we didn't want to work too hard.
- Frequent patch updates.
  - Which means binary updates.
- DMCA considerations (WoW Glider).
  - We wanted to steer away from anything that could be considered a circumvention device by restricting our code to LUA using the World of Warcraft API.
- We used Autohotkeys because it was allowed for multi-boxing.
- Later we wrote something similar for the MAC.

Binding Keys Out of Combat

```
--
-- How to use ProphetKeyBindings() in
-- our healing function.
--

function PriestHeal()
    --
    -- Binds spells, inventory items,
    -- container items, targeting,
    -- movement, and some macros to
    -- key presses.
    --
    ProphetKeyBindings();

    for i = 1, GetNumPartyMembers() do
        nextPlayer = "party" .. i;

        if (UnitHealth(nextPlayer) /
            UnitHealthMax(nextPlayer) <=
            .25)
            TargetUnit(nextPlayer);
            CastSpellByName(“Flash Heal”);
        end
    end
end
```

```
--
-- Simple example of how
-- ProphetKeyBindings() works.
--

function ProphetKeyBindings()
  --
  -- SNIP (only bind out of combat)
  --

  btn = CreateFrame("Button”,"BtpButton" ..
    k,nil,"SecureActionButtonTemplate");
  btn:RegisterForClicks("AnyUp");
  btn:SetAttribute("type”,"macro");
  btn:SetAttribute("macrotext”,
    "/focus target\n/target player");

  SetBindingClick(key .. letters[j],
    "BtpButton" .. k);
  fuckBlizMapping["player"] = key ..
    letters[j];
end
```

Map Key Binding to a Color

```
--
-- Simple example of how
-- ProphetKeyBindings() works.
--

function ProphetKeyBindings()
  --
  -- SNIP (only bind out of combat)
  --

  btn = CreateFrame("Button”,"BtpButton" ..
    k,nil,"SecureActionButtonTemplate");
  btn:RegisterForClicks("AnyUp");
  btn:SetAttribute("type”,"macro");
  btn:SetAttribute("macrotext”,
    "/focus target\n/target player");

  SetBindingClick(key .. letters[j],
    "BtpButton" .. k);
  fuckBlizMapping["player"] = key ..
    letters[j];
end
```

```
--
-- Simple example of how
-- color-to-key binding works.
--

while true do
    keyToColor[key .. letters[i]] = hex[r] ..
        hex[g] .. hex[b];
    i = i + 1;

    if (i > 45 or
       (i > 36 and key == "CTRL-")) then
        if (key == "CTRL-") then
            key = "CTRL-SHIFT-";
        elseif (key == "CTRL-SHIFT-") then
            key = "ALT-";
        elseif (key == "ALT-") then
            key = "ALT-SHIFT-";
        elseif (key == "ALT-SHIFT-") then
            key = "ALT-CTRL-";
        elseif (key == "ALT-CTRL-") then
            key = "ALT-CTRL-SHIFT-";
        elseif (key == "ALT-CTRL-SHIFT-") then
            break;
        end

        i = 1;
    end

    r, g, b = RollRGB(r, g, b);
end
```

Display Colors in Frames

```
--
-- Code to Set a Frame Color
--

function btp_frame_set_color_hex(fname, hex)
    if (fname and hex) then
        local rhex, ghex, bhex = string.sub(hex, 1, 2),
                                 string.sub(hex, 3, 4),
                                 string.sub(hex, 5, 6);

        btp_frame_set_color(fname, tonumber(rhex, 16)/255,
                                   tonumber(ghex, 16)/255,
                                   tonumber(bhex, 16)/255);
    end
end

function btp_frame_set_color(fname, red, green, blue)
    local full_name = "btp_frame_" .. fname;
    local frame     = getglobal(full_name);

    if (frame and red and green and blue) then
        frame:SetBackdropColor(red,green,blue);
    end
end
```

Replace The API's Old Functions

```
--
-- Simple example of how
-- FuckBlizzardByName() works.
--

function FuckBlizzardByName(cmd)
  if (fuckBlizMapping[cmd]) then
    btp_frame_set_color_hex("PA”,
      keyToColor[fuckBlizMapping[cmd]]);
  end

  btp_frame_set_color_hex("IT", "FFFFFF");
end

--
-- Simple example of how
-- FuckBlizzardTargetUnit() works
--
function FuckBlizzardTargetUnit(unit_id)
  if (fuckBlizMapping[unit_id]) then
    if (unit_id == "playertarget") then
      btp_frame_set_color_hex("PPT”,
        keyToColor[fuckBlizMapping[unit_id]]);
    else
      btp_frame_set_color_hex("PT",
        keyToColor[fuckBlizMapping[unit_id]]);
    end
  end

  btp_frame_set_color_hex("IT", "FFFFFF");
end
```

```
--
-- Our new healing function
--

function PriestHeal()
  for i = 1, GetNumPartyMembers() do
    nextPlayer = "party" .. i;

    if (UnitHealth(nextPlayer) <= .25)
      FuckBlizzardTargetUnit(nextPlayer);
      FuckBlizzardByName(“Flash Heal”);
    end
  end
end
```

Frames Have Context

```
--
-- Init Frames With Gradient Blue
--

function btp_frame_init()
    btp_frame_set_color_hex("IT",  "000011");
    btp_frame_set_color_hex("IA",  "000022");
    btp_frame_set_color_hex("IPT", "000033");
    btp_frame_set_color_hex("CT",  "000044");
    btp_frame_set_color_hex("CA",  "000055");
    btp_frame_set_color_hex("CPT", "000066");
    btp_frame_set_color_hex("AT",  "000077");
    btp_frame_set_color_hex("AA",  "000088");
    btp_frame_set_color_hex("APT", "000099");
    btp_frame_set_color_hex("PT",  "0000AA");
    btp_frame_set_color_hex("MA",  "0000BB");
    btp_frame_set_color_hex("PA",  "0000CC");
    btp_frame_set_color_hex("MP",  "0000DD");
    btp_frame_set_color_hex("PPT", "0000EE");
end
```

Outside Controller Programs

> Scans for gradient blue to identify each frame.
> Frame to the far right signals there's input.
> Scrape color from frame buffer for known positions.
> Hit the key sequence associated with that color.
> Controller for PC and MAC.

PC Controller: Autohotkeys

> Blizzard says dual-boxing is okay.
> Scripting language.
> API to get pixel colors.
> Reads pixels for a frame.
> Takes an action based on a color.

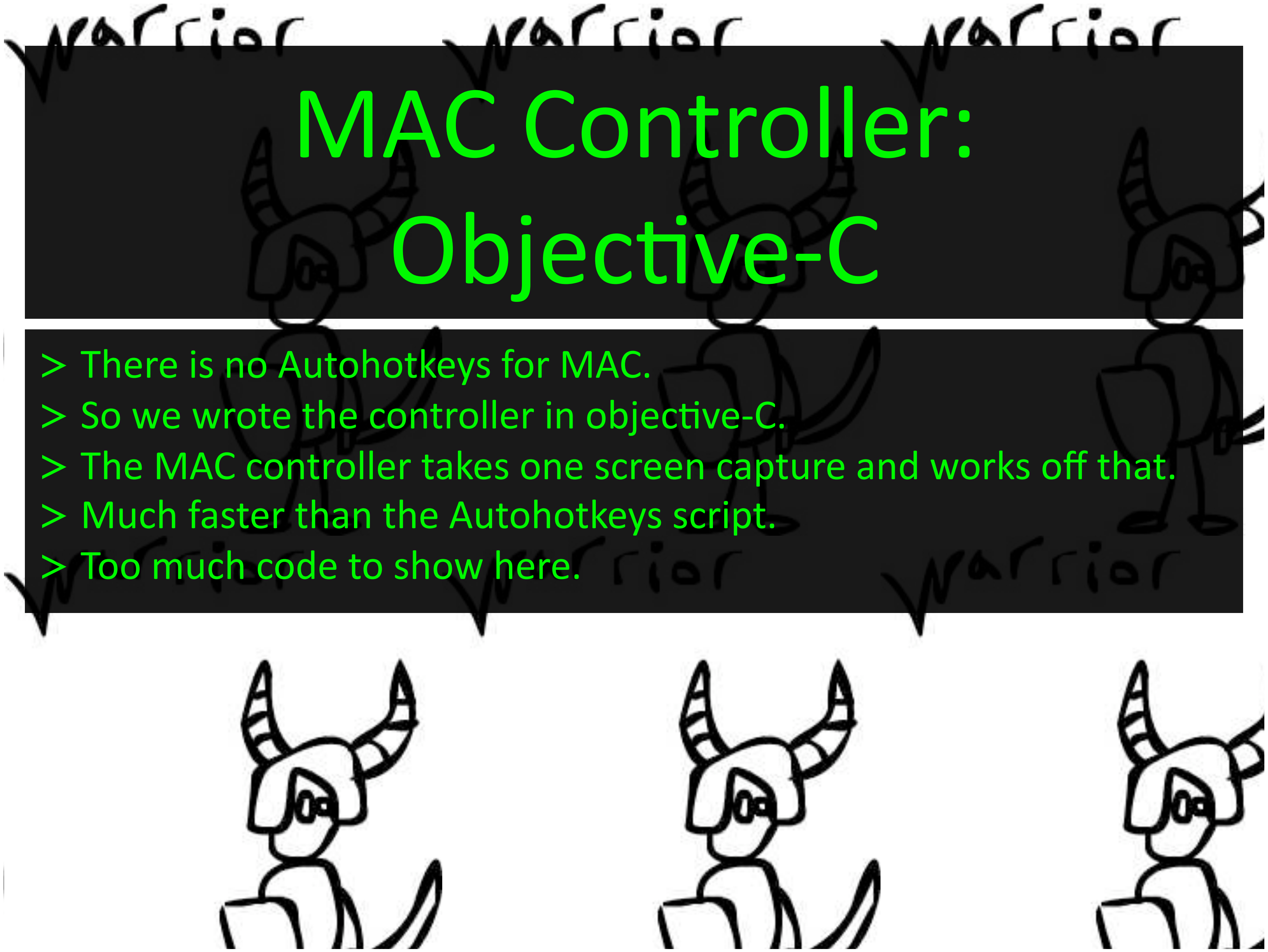
```
--
-- Example of getting a pixel
--

PixelGetColor, OutputVar, xbox, ybox, RGB

If (OutputVar == "0x000011")
{
    ...
}

--
-- Example of sending input
--

str := "focus target{Enter}”
Send /
Sleep, 75
SendInput %str%
```

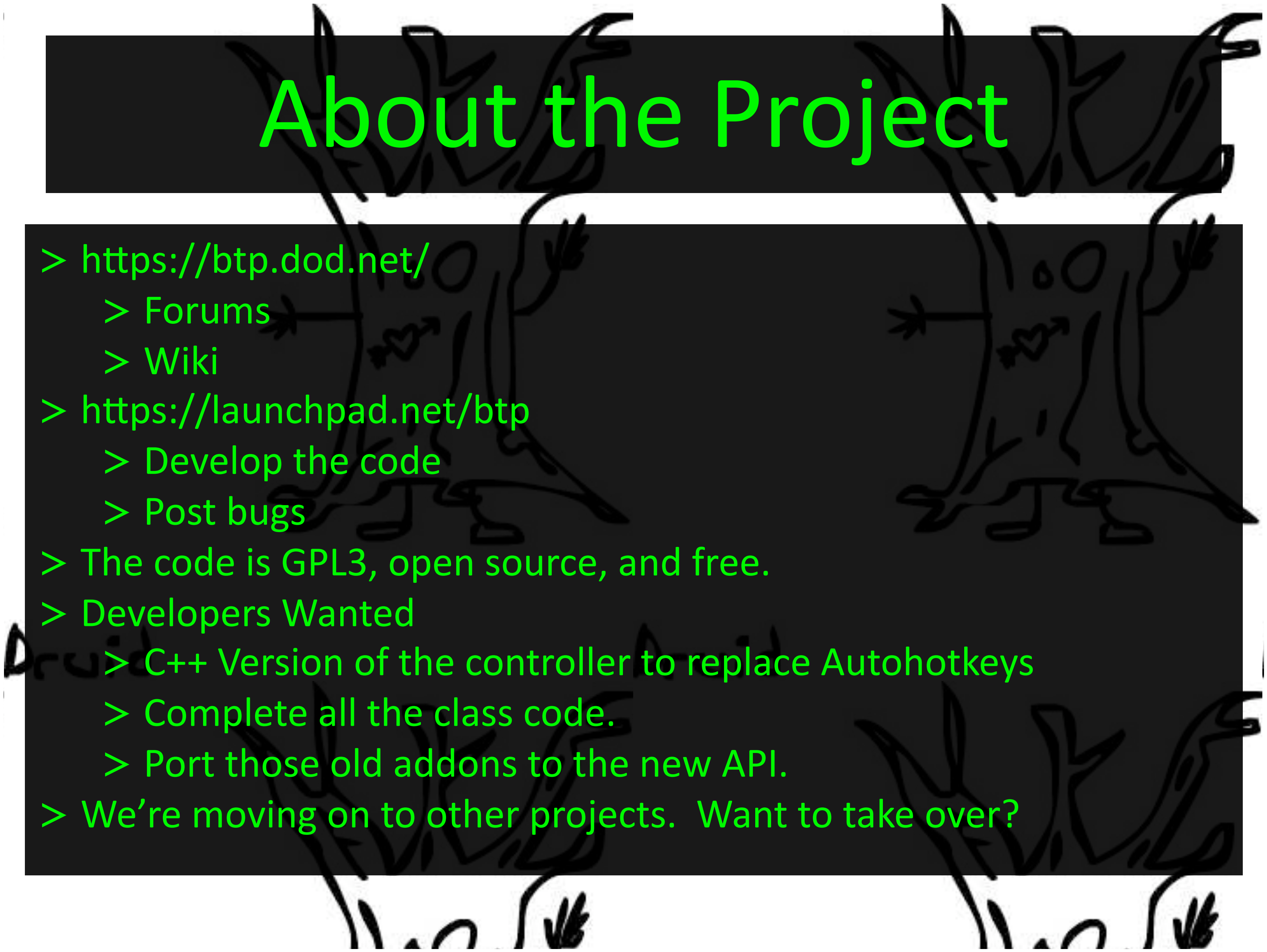


MAC Controller: Objective-C

> There is no Autohotkeys for MAC.
> So we wrote the controller in objective-C.
> The MAC controller takes one screen capture and works off that.
> Much faster than the Autohotkeys script.
> Too much code to show here.

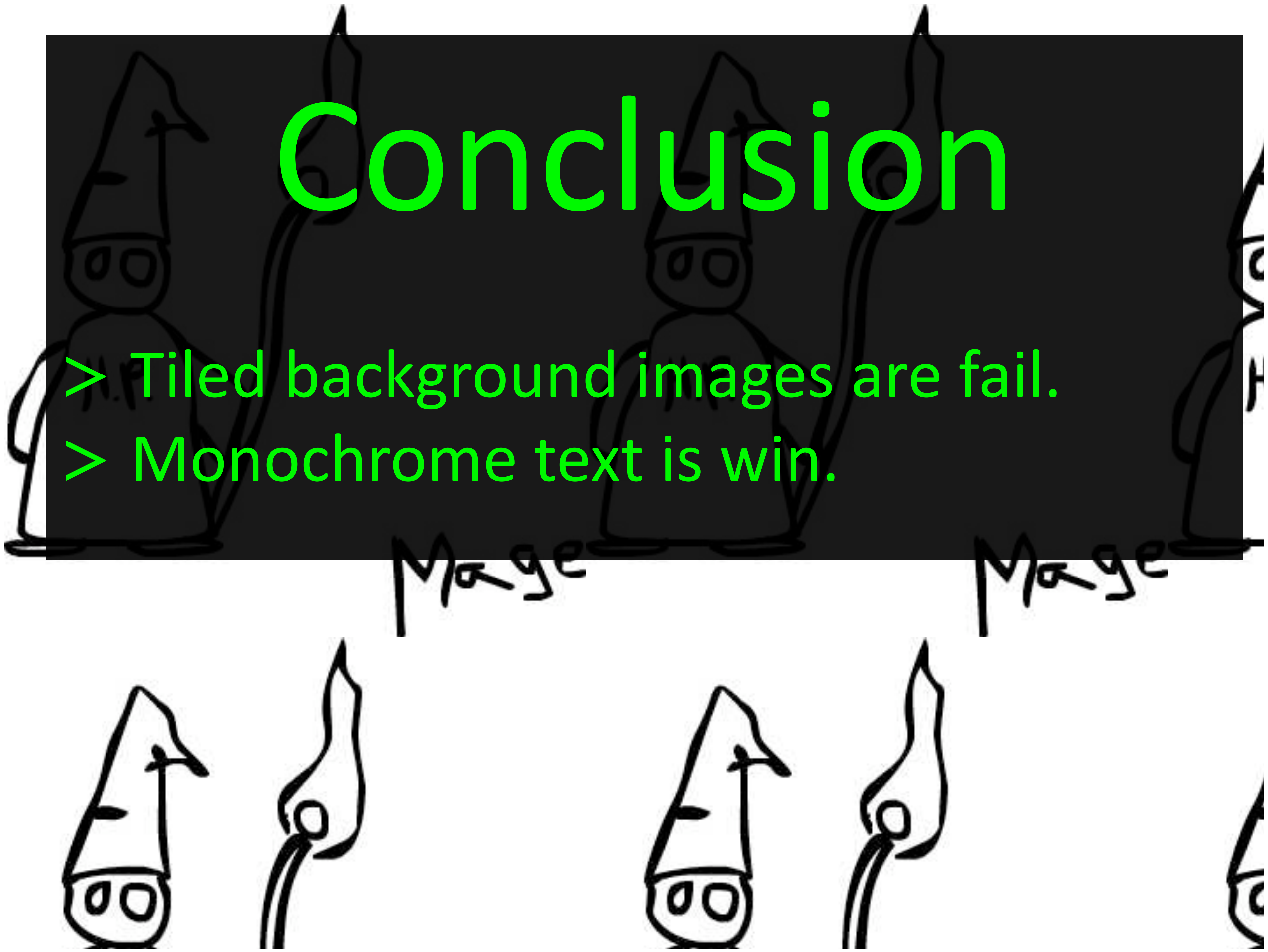
Cool Things We Can Do

> Cast spells, target, and move again.
> We can re-enable a bunch of old addons.
> We've written substitutions for one-hit-wonder and decursive.
> Heuristic casting modifications per-class.
> Cast, target, or move in response to non-hardware events.
> Healing and DPS bots that will follow another character around.
> Using the movement functions one can call a bot to them.
> Using the movement function one can farm nodes.
> Using the follow functions one can farm battlegrounds.
> A controlling player can command another to cast spells.



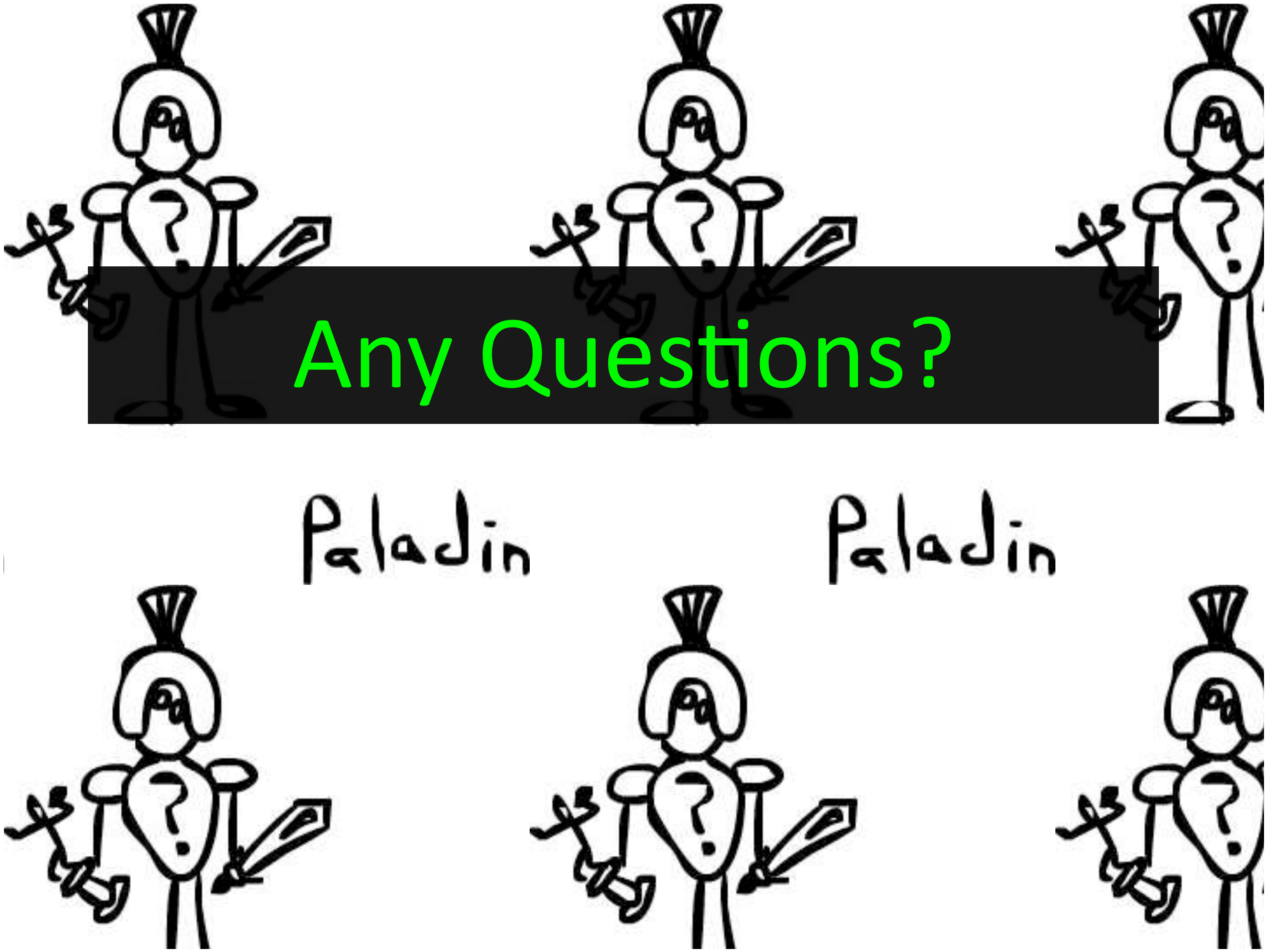
About the Project

- > https://btp.dod.net/
  - > Forums
  - > Wiki
- > https://launchpad.net/btp
  - > Develop the code
  - > Post bugs
- > The code is GPL3, open source, and free.
- > Developers Wanted
  - > C++ Version of the controller to replace Autohotkeys
  - > Complete all the class code.
  - > Port those old addons to the new API.
- > We're moving on to other projects. Want to take over?



Conclusion

> Tiled background images are fail.
> Monochrome text is win.



Any Questions?