**项目前1500行**

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Reflection;

using Ensage;

using Ensage.Common;

using Ensage.Common.Extensions;

using Ensage.Common.Extensions.Damage;

using Ensage.Common.Menu;

using Ensage.Common.Objects;

using Ensage.Common.Objects.UtilityObjects;

using Ensage.Heroes;

using log4net;

using PlaySharp.Toolkit.Logging;

using SharpDX;

namespace MeepoAnnihilation

{

internal static class Program

{

#region Members

private static readonly Menu Menu = new Menu("调度", "二次排程", true, "npc\_dota\_hero\_meepo", true);

/\*new Menu("Aniihilation", "meepo", true,"npc\_dota\_hero\_meepo");\*/

private static readonly Dictionary<Vector3, string> LaneDictionary = new Dictionary<Vector3, string>()

{

{new Vector3(-6080, 5805, 384), "top"},

{new Vector3(-6600, -3000, 384), "top"},

{new Vector3(2700, 5600, 384), "top"},

{new Vector3(5807, -5785, 384), "bot"},

{new Vector3(-3200, -6200, 384), "bot"},

{new Vector3(6200, 2200, 384), "bot"},

{new Vector3(-600, -300, 384), "middle"},

{new Vector3(3600, 3200, 384), "middle"},

{new Vector3(-4400, -3900, 384), "middle"}

};

private static readonly ILog Log = AssemblyLogs.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

private static Hero MyHero { get; set; }

private static Player MyPlayer { get; set; }

private static Hero \_globalTarget;

private static Item \_blink, \_meka, \_aghainim, \_hex, \_orchid, \_eb;

private static Ability \_ultimate;

private static List<Meepo> \_meepoList = new List<Meepo>();

public static readonly Dictionary<uint, OrderState> OrderStates = new Dictionary<uint, OrderState>();

public static readonly Dictionary<uint, OrderState> LastOrderStates = new Dictionary<uint, OrderState>();

private static readonly Dictionary<uint, ParticleEffect> Effects = new Dictionary<uint, ParticleEffect>();

public enum OrderState

{

Idle,

Jungle,

Stacking,

Laning,

Escape,

InCombo

}

private static Vector2 IconPos

=> new Vector2(Menu.Item("Drawing.posX").GetValue<Slider>().Value, Menu.Item("Drawing.posY").GetValue<Slider>().Value);

private static int IconSize => Menu.Item("Drawing.Size").GetValue<Slider>().Value;

private static readonly bool[] MenuIsOpen = new bool[10];

private static int \_selectedId;

private static List<Entity> \_selectedMeepo = new List<Entity>();

public static readonly Dictionary<uint, Ability> SpellQ = new Dictionary<uint, Ability>();

public static readonly Dictionary<uint, Ability> SpellW = new Dictionary<uint, Ability>();

private static bool ThrowingNet => Menu.Item("hotkey.ThrowNet").GetValue<KeyBind>().Active;

private static bool isNetEnable = false;

#endregion

#region Main

private static bool \_firstTime = true;

private static void Main()

{

Events.OnLoad += (sender, args) =>

{

InitMenu();

MyHero = ObjectManager.LocalHero;

if (MyHero.HeroId != HeroId.npc\_dota\_hero\_meepo)

return;

Game.PrintMessage(

"<font face='Comic Sans MS, cursive'><font color='#00aaff'>" + Menu.DisplayName + " By Jumpering" +

" loaded!</font> <font color='#aa0000'>v" + Assembly.GetExecutingAssembly().GetName().Version);

Game.OnUpdate += Game\_OnUpdate;

Game.OnUpdate += Camp\_update;

Drawing.OnDraw += Drawing\_OnDraw;

Game.OnWndProc += Game\_OnWndProc;

Player.OnExecuteOrder += Player\_OnExecuteAction;

Orbwalking.Load();

\_meepoList.Clear();

\_aghainim = null;

\_blink = null;

\_meka = null;

\_ultimate = null;

JungleCamps.Init();

foreach (var camp in JungleCamps.GetCamps)

{

camp.CanBeHere = true;

}

\_selectedMeepo.Clear();

\_meepoList.Clear();

MeepoSet.Clear();

ScaleX = Drawing.Width / 1920f;

ScaleY = Drawing.Height / 1080f;

};

Events.OnClose += (sender, args) =>

{

Game.OnUpdate -= Game\_OnUpdate;

Game.OnUpdate -= Camp\_update;

Drawing.OnDraw -= Drawing\_OnDraw;

Game.OnWndProc -= Game\_OnWndProc;

Player.OnExecuteOrder -= Player\_OnExecuteAction;

MyHero = null;

};

}

public static float ScaleY { get; set; }

public static float ScaleX { get; set; }

public static void InitMenu()

{

if (!\_firstTime)

return;

\_firstTime = false;

Menu.AddItem(new MenuItem("Enable", "Enable").SetValue(true));

Menu.AddItem(new MenuItem("hotkey", "Hotkey").SetValue(new KeyBind('G', KeyBindType.Press)));

Menu.AddItem(

new MenuItem("hotkey.PoofAll", "Poof all to selected meepo").SetValue(new KeyBind('Q', KeyBindType.Press)));

Menu.AddItem(new MenuItem("LockTarget", "LockTarget").SetValue(true));

Menu.AddItem(

new MenuItem("hotkey.Escape", "Escape for selected Meepo(s)").SetValue(new KeyBind('V',

KeyBindType.Press)));

Menu.AddItem(

new MenuItem("hotkey.ThrowNet", "Throw net").SetValue(new KeyBind('D',

KeyBindType.Press))).ValueChanged += (sender, args) =>

{

var newValue = args.GetNewValue<KeyBind>().Active;

var oldValue = args.GetOldValue<KeyBind>().Active;

if (oldValue != newValue)

isNetEnable = newValue;

};

Menu.AddItem(new MenuItem("Escape.MinRange", "Min health for autoheal").SetValue(new Slider(300, 0, 4000)));

Menu.AddItem(

new MenuItem("Escape.MinRangePercent", "Min health for autoheal (%)").SetValue(new Slider(15, 0, 100)));

Menu.AddItem(

new MenuItem("Drawing.PoffSystem", "Poofer").SetValue(true)

.SetTooltip("All selected meepos will use W spell on target position when first meepo use W spell"));

var drawingMenu = new Menu("Drawing", "Drawing");

drawingMenu.AddItem(new MenuItem("Drawing.DamageFromPoof", "Draw Poof count on enemy").SetValue(true));

drawingMenu.AddItem(new MenuItem("Drawing.NumOfMeepo", "Draw Number for each meepo").SetValue(true));

drawingMenu.AddItem(

new MenuItem("Drawing.NumOfMeepoOnMinimap", "Draw Number for each meepo on minimap").SetValue(true));

drawingMenu.AddItem(

new MenuItem("Drawing.NumOfMeepoInMenu", "Draw Number for each meepo in OverlayMenu").SetValue(true));

drawingMenu.AddItem(

new MenuItem("Drawing.posX", "OverlayMenu pos X").SetValue(new Slider(120, 0,

500)));

drawingMenu.AddItem(

new MenuItem("Drawing.posY", "OverlayMenu pos Y").SetValue(new Slider(137, 0,

500)));

drawingMenu.AddItem(

new MenuItem("Drawing.Size", "OverlayMenu size").SetValue(new Slider(75, 0,

500)));

drawingMenu.AddItem(new MenuItem("Drawing.PoofState", "Draw poof state in OverlayMenu").SetValue(true));

var autoPush = new Menu("Auto Push", "AutoPush");

autoPush.AddItem(

new MenuItem("AutoPush.Enable", "Push Lane By Selected Meepo(s)").SetValue(new KeyBind('Z',

KeyBindType.Press)));

autoPush.AddItem(

new MenuItem("AutoPush.EscapeFromAnyEnemyHero", "Escape From Enemy Hero in selected range").SetValue(

true));

autoPush.AddItem(new MenuItem("AutoPush.EscapeRange", "Range For Escape").SetValue(new Slider(1500, 0, 5000)));

autoPush.AddItem(new MenuItem("AutoPush.AutoW", "use w for lasthitting").SetValue(true));

autoPush.AddItem(new MenuItem("AutoPush.TravelBoots", "use TravelBoots").SetValue(false));

autoPush.AddItem(new MenuItem("AutoPush.LastHitMode", "Try to last hit creeps").SetValue(true));

var jumgleFarm = new Menu("Jungle Farm", "JungleFarm");

jumgleFarm.AddItem(

new MenuItem("JungleFarm.Enable", "Jungle Farm By Selected Meepo(s)").SetValue(new KeyBind('X',

KeyBindType.Press)));

jumgleFarm.AddItem(

new MenuItem("JungleFarm.EscapeFromAnyEnemyHero", "Escape From Enemy Hero in selected range").SetValue(

true));

jumgleFarm.AddItem(

new MenuItem("JungleFarm.EscapeRange", "Range For Escape").SetValue(new Slider(1500, 0, 5000)));

jumgleFarm.AddItem(new MenuItem("JungleFarm.AutoW", "use w for farming").SetValue(true));

jumgleFarm.AddItem(new MenuItem("JungleFarm.TeamCheck", "Farm enemy jungle too").SetValue(false));

jumgleFarm.AddItem(new MenuItem("JungleFarm.Ancient", "Farm ancients too").SetValue(false));

var jungleStack = new Menu("Jungle Stack", "JungleStack");

jungleStack.AddItem(

new MenuItem("JungleStack.Enable", "Jungle Stacking By Selected Meepo(s)").SetValue(new KeyBind('C',

KeyBindType.Press)));

jungleStack.AddItem(

new MenuItem("JungleStack.EscapeFromAnyEnemyHero", "Escape From Enemy Hero in selected range").SetValue(

true));

jungleStack.AddItem(

new MenuItem("JungleStack.EscapeRange", "Range For Escape").SetValue(new Slider(1500, 0, 5000)));

jungleStack.AddItem(new MenuItem("JungleStack.TeamCheck", "Stack enemy jungle too").SetValue(false));

jungleStack.AddItem(new MenuItem("JungleStack.Ancient", "Stack ancients too").SetValue(false));

Menu.AddSubMenu(autoPush);

Menu.AddSubMenu(jumgleFarm);

Menu.AddSubMenu(jungleStack);

Menu.AddSubMenu(drawingMenu);

Menu.AddToMainMenu();

}

#endregion

#region ActionsWithLocalPlayer

private static void Player\_OnExecuteAction(Player sender, ExecuteOrderEventArgs args)

{

if (!Menu.Item("Enable").GetValue<bool>()) return;

if (!args.IsPlayerInput)

return;

if (MyHero == null || !MyHero.IsValid || !MyHero.IsAlive) return;

var me = sender.Selection.First();

var order = args.OrderId;

if (order == OrderId.Hold || order == OrderId.MoveLocation)

{

foreach (

var me2 in

args.Entities.Select(entity => MeepoSet.Find(x => x.Handle == entity.Handle))

.Where(me2 => me2 != null)

.Where(me2 => me2.CurrentOrderState != OrderState.Escape))

{

OrderStates[me2.Handle] = OrderState.Idle;

}

}

else if (Menu.Item("Drawing.PoffSystem").GetValue<bool>() &&

(args.OrderId == OrderId.AbilityLocation || args.OrderId == OrderId.AbilityTarget) &&

args.Ability.StoredName() == SpellW[MyHero.Handle].Name)

{

var pos = args.TargetPosition;

foreach (var meepo in \_selectedMeepo.Where(x => !Equals(x, me)))

{

var handle = meepo.Handle;

var spell = SpellW[handle];

if (spell == null)

{

continue;

}

if (spell.CanBeCasted())

{

spell.UseAbility(pos);

}

}

}

}

private static void Game\_OnWndProc(WndEventArgs args)

{

if (!Menu.Item("Enable").GetValue<bool>()) return;

if (MyHero == null || !MyHero.IsValid || !MyHero.IsAlive) return;

if (args.WParam != 1 || Game.IsChatOpen)

{

\_leftMouseIsPress = false;

return;

}

\_leftMouseIsPress = true;

}

#endregion

#region Update & Draw

private static void Drawing\_OnDraw(EventArgs args)

{

/\*var ss = SpellW[MyHero.Handle];

var s = string.Format("materials/ensage\_ui/spellicons/{0}.vmat", ss.StoredName());

var pos = WorldToMinimap(MyHero.Position);

Drawing.DrawRect(

pos,

new Vector2(20, 20),

Textures.GetTexture(s));\*/

if (!Menu.Item("Enable").GetValue<bool>()) return;

try

{

if (Menu.Item("Drawing.DamageFromPoof").GetValue<bool>() && SpellW[MyHero.Handle] != null)

{

var poof = SpellW[MyHero.Handle];

foreach (

var hero in

Heroes.GetByTeam(MyHero.GetEnemyTeam()).Where(x => x.IsValid && x.IsAlive && x.IsVisible))

{

var w2SPos = HUDInfo.GetHPbarPosition(hero);

if (w2SPos.X > 0 && w2SPos.Y > 0)

{

var sizeX = HUDInfo.GetHPBarSizeX();

var sizeY = HUDInfo.GetHpBarSizeY();

var handle = hero.Handle;

var damagePerPoof = Calculations.DamageTaken(hero,

poof.GetDamage(poof.Level), DamageType.Magical, MyHero);

var minCounter = (int) (hero.Health/damagePerPoof);

var count = ((minCounter == int.MinValue) ? "Invul" : minCounter.ToString());

var textSize = Drawing.MeasureText(count, "Arial",

new Vector2((float) (sizeY\*1.5), 500), FontFlags.AntiAlias);

var textPos = w2SPos - new Vector2(textSize.X + 5, (float) ((sizeY\*1.5) - (textSize.Y)));

Drawing.DrawText(

count,

textPos,

new Vector2((float) (sizeY\*1.5), 100),

Color.White,

FontFlags.AntiAlias | FontFlags.StrikeOut);

var texturename = $"materials/ensage\_ui/spellicons/{poof.StoredName()}.vmat";

var iconPos = textPos - new Vector2(sizeY\*2 + 5, 0);

Drawing.DrawRect(

iconPos,

new Vector2(sizeY\*2, sizeY\*2),

Textures.GetTexture(texturename));

}

}

}

}

catch

{

// ignored

}

if (true)

{

foreach (var meepo in MeepoSet/\*.OrderBy(x=>x.Id)\*/)

{

var handle = meepo.Handle;

/\*Drawing.DrawRect(IconPos + new Vector2(0, IconSize)\*count++, new Vector2(20, 50),

new Color(0, 155, 255, 155));

Drawing.DrawText(OrderStates[handle].ToString(), IconPos + new Vector2(5, IconSize) \* (count-1),

new Vector2(20, 5), Color.Red, FontFlags.AntiAlias | FontFlags.Additive | FontFlags.Custom);\*/

if (Menu.Item("Drawing.NumOfMeepoInMenu").GetValue<bool>())

{

var sizeY = HUDInfo.GetHpBarSizeY();

var pos = IconPos + new Vector2(0, IconSize)\* meepo.Id;

var textSize = Drawing.MeasureText((meepo.Id+1).ToString(CultureInfo.InvariantCulture), "Arial",

new Vector2((float) (sizeY\*3), 100), FontFlags.AntiAlias);

var textPos = pos - new Vector2(sizeY / 2 + textSize.Y, 0);

Drawing.DrawText(

(meepo.Id+1).ToString(CultureInfo.InvariantCulture),

textPos - new Vector2(72, 0),

new Vector2((float) (sizeY\*3), 100),

Color.White,

FontFlags.AntiAlias | FontFlags.StrikeOut);

}

DrawButton(IconPos + new Vector2(0, IconSize)\* meepo.Id, 70, 30, ref MenuIsOpen[meepo.Id],

new Color(0, 155, 255, 150), new Color(0, 0, 0, 100), OrderStates[handle].ToString(),

\_selectedMeepo.Contains(meepo.Hero));

if (Menu.Item("Drawing.PoofState").GetValue<bool>())

{

var w = meepo.SpellW;

if (w.IsInAbilityPhase)

{

var delta = (float) ((Game.RawGameTime - meepo.PoofStartTime)\*70/1.5);

Drawing.DrawRect(IconPos + new Vector2(0, 32 + (IconSize) \* meepo.Id), new Vector2(delta, 10),

Color.White);

Drawing.DrawRect(IconPos + new Vector2(0, 32 + (IconSize) \* meepo.Id), new Vector2(70, 10),

new Color(0,0,0,100));

Drawing.DrawRect(IconPos + new Vector2(0, 32 + (IconSize) \* meepo.Id), new Vector2(70, 10),

Color.Black, true);

}

else

{

var state = w.AbilityState;

var clr = state == AbilityState.NotEnoughMana

? new Color(0, 155, 255, 120)

: state == AbilityState.Ready

? new Color(0, 0, 0, 0)

: new Color(255, 50, 50, 120);

Drawing.DrawRect(IconPos + new Vector2(0, 32 + (IconSize)\*meepo.Id), new Vector2(20, 20),

Textures.GetSpellTexture(w.StoredName()));

Drawing.DrawRect(IconPos + new Vector2(0, 32 + (IconSize)\*meepo.Id), new Vector2(20, 20),

clr);

Drawing.DrawRect(IconPos + new Vector2(0, 32 + (IconSize)\*meepo.Id), new Vector2(20, 20),

Color.Black, true);

}

}

if (MenuIsOpen[meepo.Id])

{

\_selectedId = 0;

DrawButton(IconPos + new Vector2(0, IconSize)\* meepo.Id + new Vector2(70, 0), 70, 30, ref \_selectedId,

1,

new Color(0, 0, 0, 100),

OrderState.Idle.ToString());

DrawButton(IconPos + new Vector2(0, IconSize)\* meepo.Id + new Vector2(70, 30), 70, 30,

ref \_selectedId, 2,

new Color(0, 0, 0, 100),

OrderState.Jungle.ToString());

DrawButton(IconPos + new Vector2(0, IconSize)\* meepo.Id + new Vector2(140, 0), 70, 30,

ref \_selectedId, 3,

new Color(0, 0, 0, 100),

OrderState.Stacking.ToString());

DrawButton(IconPos + new Vector2(0, IconSize)\* meepo.Id + new Vector2(140, 30), 70, 30,

ref \_selectedId, 4,

new Color(0, 0, 0, 100),

OrderState.Laning.ToString());

DrawButton(IconPos + new Vector2(0, IconSize)\* meepo.Id + new Vector2(71 + 70/2, 60), 70, 30,

ref \_selectedId, 5,

new Color(0, 0, 0, 100),

OrderState.Escape.ToString());

if (\_selectedId != 0)

{

OrderStates[handle] = (OrderState) \_selectedId - 1;

MenuIsOpen[meepo.Id] = false;

if (OrderStates[handle] == OrderState.Escape)

{

NeedHeal[handle] = true;

}

}

}

if (Menu.Item("Drawing.NumOfMeepo").GetValue<bool>())

{

var w2SPos = HUDInfo.GetHPbarPosition(meepo.Hero);

if (w2SPos.X > 0 && w2SPos.Y > 0)

{

var sizeX = HUDInfo.GetHPBarSizeX();

var sizeY = HUDInfo.GetHpBarSizeY();

var text = meepo.Id+1;

var textSize = Drawing.MeasureText(text.ToString(CultureInfo.InvariantCulture), "Arial",

new Vector2((float)(sizeY \* 3), 100), FontFlags.AntiAlias);

var textPos = w2SPos + new Vector2(sizeY / 2 + textSize.Y, 0);

Drawing.DrawText(

text.ToString(CultureInfo.InvariantCulture),

textPos + new Vector2(0, -50),

new Vector2((float)(sizeY \* 3), 100),

Color.White,

FontFlags.AntiAlias | FontFlags.StrikeOut);

}

}

if (Menu.Item("Drawing.NumOfMeepoOnMinimap").GetValue<bool>())

{

var w2SPos = meepo.Hero.Position.WorldToMinimap();//WorldToMinimap(meepo.NetworkPosition);

var sizeY = HUDInfo.GetHpBarSizeY();

var text = meepo.Id+1;

Drawing.DrawText(

text.ToString(CultureInfo.InvariantCulture),

w2SPos + new Vector2(-5, -33),

new Vector2((float)(sizeY \* 3), 100),

Color.White,

FontFlags.AntiAlias | FontFlags.StrikeOut);

}

}

}

}

private static void Game\_OnUpdate(EventArgs args)

{

if (!Menu.Item("Enable").GetValue<bool>()) return;

if (MyHero == null || !MyHero.IsValid)

{

MyHero = ObjectManager.LocalHero;

if (MyHero != null)

Print("[Informer]: Main Hero was found");

return;

}

if (MyPlayer == null || !MyPlayer.IsValid)

{

MyPlayer = ObjectManager.LocalPlayer;

if (MyPlayer != null)

Print("[Informer]: Player was found");

return;

}

if (\_aghainim == null || !\_aghainim.IsValid)

{

\_aghainim = MyHero.FindItem("item\_ultimate\_scepter");

if (\_aghainim != null) Print("[Informer]: aghainim was found");

}

if (\_blink == null || !\_blink.IsValid)

{

\_blink = MyHero.FindItem("item\_blink");

if (\_blink != null) Print("[Informer]: blink was found");

}

if (\_meka == null || !\_meka.IsValid)

{

\_meka = MyHero.FindItem("item\_mekansm");

if (\_meka != null) Print("[Informer]: mekansm was found");

}

if (\_eb == null || !\_eb.IsValid)

{

\_eb = MyHero.FindItem("item\_ethereal\_blade");

if (\_eb != null) Print("[Informer]: ethereal\_blade was found");

}

if (\_hex == null || !\_hex.IsValid)

{

\_hex = MyHero.FindItem("item\_sheepstick");

if (\_hex != null) Print("[Informer]: hex was found");

}

if (\_orchid == null || !\_orchid.IsValid)

{

\_orchid = MyHero.FindItem("item\_orchid");

if (\_orchid != null) Print("[Informer]: orchid was found");

}

if (\_ultimate == null || !\_ultimate.IsValid)

{

\_ultimate = MyHero.Spellbook.Spell4;

if (\_ultimate != null) Print("[Informer]: ultimate was found");

}

/\*if (travelBoots == null || !travelBoots.IsValid)

{

travelBoots = MyHero.FindItem("item\_travel\_boots") ?? MyHero.FindItem("item\_travel\_boots\_2");

if (travelBoots != null) Print("[Informer]: TravelBoots was found");

}\*/

if (MyHero == null || !MyHero.IsValid || !MyHero.IsAlive) return;

if (Menu.Item("AutoPush.Enable").GetValue<KeyBind>().Active && Utils.SleepCheck("button\_cd"))

{

foreach (var handle in \_selectedMeepo.Select(x => x.Handle))

{

if (OrderStates[handle] == OrderState.Laning)

OrderStates[handle] = OrderState.Idle;

else

OrderStates[handle] = OrderState.Laning;

}

Utils.Sleep(250, "button\_cd");

}

if (Menu.Item("JungleFarm.Enable").GetValue<KeyBind>().Active && Utils.SleepCheck("button\_cd"))

{

foreach (var handle in \_selectedMeepo.Select(x => x.Handle))

{

if (OrderStates[handle] == OrderState.Jungle)

OrderStates[handle] = OrderState.Idle;

else

OrderStates[handle] = OrderState.Jungle;

}

Utils.Sleep(250, "button\_cd");

}

if (Menu.Item("JungleStack.Enable").GetValue<KeyBind>().Active && Utils.SleepCheck("button\_cd"))

{

foreach (var me in \_selectedMeepo)

{

var handle = me.Handle;

if (OrderStates[handle] == OrderState.Stacking)

{

var s = JungleCamps.GetCamps.Find(x => Equals(x.Stacking, me));

if (s != null)

{

s.Stacking = null;

}

OrderStates[handle] = OrderState.Idle;

}

else

OrderStates[handle] = OrderState.Stacking;

}

Utils.Sleep(250, "button\_cd");

}

RefreshMeepoList();

foreach (var meepo in MeepoSet)

{

var w = meepo.SpellW;

if (w.IsInAbilityPhase)

{

if (meepo.PoofStartTime == float.MaxValue)

{

meepo.PoofStartTime = Game.RawGameTime;

}

}

else if (meepo.PoofStartTime != float.MaxValue)

{

meepo.PoofStartTime = float.MaxValue;

}

SafeTp(meepo.Hero, meepo.SpellW);

}

if (Menu.Item("hotkey.Escape").GetValue<KeyBind>().Active && Utils.SleepCheck("button\_cd"))

{

foreach (var handle in \_selectedMeepo.Select(me => me.Handle))

{

if (OrderStates[handle] == OrderState.Escape)

OrderStates[handle] = OrderState.Idle;

else

{

NeedHeal[handle] = true;

OrderStates[handle] = OrderState.Escape;

}

}

Utils.Sleep(250, "button\_cd");

}

if (Menu.Item("hotkey.PoofAll").GetValue<KeyBind>().Active)

{

foreach (

var me in

MeepoSet.Where(

x =>

x.Hero.IsAlive && x.CurrentOrderState != OrderState.Escape &&

!Equals(\_selectedMeepo.FirstOrDefault(), x.Hero)))

{

var handle = me.Handle;

var spell = me.SpellW;

if (spell != null && spell.CanBeCasted() && Utils.SleepCheck("all\_poof" + handle))

{

spell.UseAbility((Unit) \_selectedMeepo.First());

Utils.Sleep(250, "all\_poof" + handle);

}

}

}

if (ThrowingNet && isNetEnable)

{

//var target = TargetSelector.ClosestToMouse(MyHero);

var target = Game.MousePosition;

if (true)

{

foreach (

var me in

MeepoSet.Where(

x =>

x.Hero.IsAlive && x.CurrentOrderState != OrderState.Escape && x.Hero.Distance2D(target)<=x.SpellQ.GetCastRange())

.OrderBy(y => y.Hero.Distance2D(target)))

{

var spell = me.SpellQ;

var handle = me.Handle;

if (spell != null && spell.CanBeCasted() && Utils.SleepCheck("throwNet"+handle))

{

spell.UseAbility(target);

isNetEnable = false;

Utils.Sleep(1000, "throwNet"+handle);

return;

}

}

}

}

if (!Menu.Item("hotkey").GetValue<KeyBind>().Active || (\_globalTarget != null && !\_globalTarget.IsAlive))

{

\_globalTarget = null;

FlushEffect();

return;

}

if (\_globalTarget == null || !\_globalTarget.IsValid || !Menu.Item("LockTarget").GetValue<bool>())

{

\_globalTarget = ClosestToMouse(MyHero, 300);

}

if (!MyHero.IsAlive) return;

if (\_globalTarget == null || !\_globalTarget.IsValid || !\_globalTarget.IsAlive) return;

DoCombo(\_globalTarget);

}

#endregion

#region MainShit

private static void DoCombo(Hero target)

{

var theClosestMeepo = \_meepoList.OrderBy(target.Distance2D).First();

var dist = theClosestMeepo.Distance2D(target)+MyHero.HullRadius+target.HullRadius;

var targetPos = target.Position;

#region GetItems&Spells

if (OrderStates[MyHero.Handle] != OrderState.Escape)

{

if (\_blink != null && \_blink.CanBeCasted() && dist <= 1150 && dist >= 250 && Utils.SleepCheck("Blink"))

{

\_blink.UseAbility(targetPos);

Utils.Sleep(250, "Blink");

}

var bkb = target.FindItem("item\_black\_king\_bar");

if (bkb != null && bkb.CanBeCasted() && \_hex != null && \_hex.CanBeCasted(target) &&

Utils.SleepCheck("hex"))

{

\_hex.UseAbility(target);

Utils.Sleep(250, "hex");

}

if (\_orchid != null && \_orchid.CanBeCasted(target) && !target.IsHexed() && Utils.SleepCheck("orchid") &&

Utils.SleepCheck("hex"))

{

\_orchid.UseAbility(target);

Utils.Sleep(250, "orchid");

}

if (\_hex != null && \_hex.CanBeCasted(target) && !target.IsSilenced() && Utils.SleepCheck("hex") &&

Utils.SleepCheck("orchid"))

{

\_hex.UseAbility(target);

Utils.Sleep(250, "hex");

}

if (\_eb != null && \_eb.CanBeCasted(target) && Utils.SleepCheck("eb"))

{

\_eb.UseAbility(target);

Utils.Sleep(250, "eb");

}

}

#endregion

foreach (

var handle in

\_meepoList.Where(x => x.IsAlive && OrderStates[x.Handle] != OrderState.Escape)

.Select(meepo => meepo.Handle))

{

OrderStates[handle] = OrderState.InCombo;

}

foreach (var meepo in \_meepoList.Where(x => x.IsAlive && OrderStates[x.Handle] == OrderState.InCombo).OrderBy(y=>y.Distance2D(target)))

{

#region gettings spells and drawing effects

var handle = meepo.Handle;

DrawEffects(meepo, target);

Ability q, w;

if (!SpellQ.TryGetValue(handle, out q))

SpellQ[handle] = meepo.Spellbook.Spell1;

if (!SpellW.TryGetValue(handle, out w))

SpellW[handle] = meepo.Spellbook.Spell2;

#region Change Orders

if (SafeTp(meepo, w))

continue;

#endregion

#endregion

#region CastQ

var mod = target.FindModifier("modifier\_meepo\_earthbind");

var remTime = mod?.RemainingTime ?? 0;

if ((\_blink==null || !\_blink.CanBeCasted()) && q != null && q.CanBeCasted() && dist <= q.CastRange &&

(mod == null || remTime <= .7) &&

Utils.SleepCheck("Period\_q"))

{

if (q.CastSkillShot(target))

//if (q.CastStun(target))

{

Utils.Sleep(q.GetHitDelay(target)\*1000+100, "Period\_q");

}

}

#endregion

#region CastW

if (w != null)

{

var castRange = w.GetRealCastRange();

if ((!Equals(theClosestMeepo, meepo) || target.IsHexed() || target.IsStunned() ||

target.MovementSpeed <= 200 || (remTime > 1.3)) && w.CanBeCasted() &&

dist <= castRange &&

Utils.SleepCheck("Period\_w" + handle))

{

w.UseAbility(theClosestMeepo);

Utils.Sleep(1500, "Period\_w" + handle);

}

if (!Utils.SleepCheck("Period\_w" + handle))

{

if (dist >= castRange)

{

Utils.Sleeps.Remove("Period\_w" + handle);

meepo.Stop();

}

}

}

#endregion

#region AutoAttack

if (!target.IsVisible)

{

if (Utils.SleepCheck("attack\_rate" + handle))

{

Utils.Sleep(250, "attack\_rate" + handle);

meepo.Move(Prediction.InFront(target, 250));

}

}

else

{

var orb = OrbWalkManager(meepo);

orb?.OrbwalkOn(target, followTarget: true);

}

#endregion

}

#region Auto Meka

if (NeedUseMeka() && \_meka != null && \_meka.CanBeCasted() && Utils.SleepCheck("meka"))

{

\_meka.UseAbility();

Utils.Sleep(250, "meka");

}

#endregion

}

private static void JungleFarm(Hero me)

{

var s = JungleCamps.FindClosestCamp(me, Menu.Item("JungleFarm.TeamCheck").GetValue<bool>(),

Menu.Item("JungleFarm.Ancient").GetValue<bool>());

string name;

var enemyHeroes = Heroes.GetByTeam(me.GetEnemyTeam()).Where(x => x.IsAlive).ToList();

var dist = Menu.Item("JungleFarm.EscapeRange").GetValue<Slider>().Value;

if (Menu.Item("JungleFarm.EscapeFromAnyEnemyHero").GetValue<bool>() &&

enemyHeroes.Any(x => x.Distance2D(me) <= dist)) //escape from hero

{

var handle = me.Handle;

OrderStates[handle] = OrderState.Escape;

NeedHeal[handle] = true;

}

if (s == null)

{

s = JungleCamps.GetCamps.OrderBy(x => x.StackPosition.Distance2D(me)).FirstOrDefault();

if (s != null)

{

name = MeepoSet.Find(x => Equals(x.Hero, me)).Handle.ToString();

if (Utils.SleepCheck("MOVIER\_jungle" + name))

{

me.Move(s.StackPosition);

Utils.Sleep(500, "MOVIER\_jungle" + name);

}

}

return;

}

name = MeepoSet.Find(x => Equals(x.Hero, me)).Handle.ToString();

var anyMeepo =

MeepoSet.Where(

x =>

x.CurrentOrderState == OrderState.Jungle && x.IsAlive && x.Handle != me.Handle &&

x.Hero.Health >= Menu.Item("Escape.MinRange").GetValue<Slider>().Value)

.OrderBy(y => y.Hero.Distance2D(me))

.FirstOrDefault();

if (anyMeepo != null && me.Health <= 500 && anyMeepo.Hero.Distance2D(MyHero) <= 400 &&

CheckForChangedHealth(me))

{

if (!Utils.SleepCheck(name + "attack\_test")) return;

Utils.Sleep(200, name + "attack\_test");

var enemy =

ObjectManager.GetEntities<Unit>()

.FirstOrDefault(

x =>

x.IsAlive && x.IsVisible && x.Team != MyHero.Team && x.Distance2D(MyHero) <= 500 &&

!x.IsWaitingToSpawn);

if (enemy != null)

{

var creep = enemy.Position;

var ang = me.FindAngleBetween(creep, true);

var p = new Vector3((float) (me.Position.X - 250\*Math.Cos(ang)),

(float) (me.Position.Y - 250\*Math.Sin(ang)), 0);

me.Move(p);

}

me.Attack(anyMeepo.Hero, true);

return;

}

var mySet = MeepoSet.Find(x => Equals(x.Hero, me));

var w = mySet.SpellW;

if (w != null && Menu.Item("JungleFarm.AutoW").GetValue<bool>() && w.CanBeCasted())

{

var enemy =

ObjectManager

.GetEntities<Unit>(

)

.FirstOrDefault(

x =>

x.IsAlive && x.Health > 80 && x.IsVisible && x.Team != MyHero.Team &&

x.Distance2D(me) <= 375 &&

!x.IsWaitingToSpawn);

if (enemy != null && Utils.SleepCheck("jungle\_farm\_w" + name))

{

w.UseAbility(enemy.Position);

Utils.Sleep(1500, "jungle\_farm\_w\_inCasting" + name);

Utils.Sleep(250, "jungle\_farm\_w" + name);

}

else if (enemy == null && !Utils.SleepCheck("jungle\_farm\_w\_inCasting" + name) &&

Utils.SleepCheck("jungle\_farm\_w\_stop" + name))

{

me.Stop();

Utils.Sleeps.Remove("jungle\_farm\_w\_inCasting" + name);

Utils.Sleep(500, "jungle\_farm\_w\_stop" + name);

}

}

if (!Utils.SleepCheck(name + "attack") || me.IsAttacking()) return;

Utils.Sleep(500, name + "attack");

me.Attack(s.CampPosition);

}

private static bool SafeTp(Unit me, Ability w)

{

if (true) //(Menu.Item("AutoHeal.Hero.Enable").GetValue<bool>())

{

var handle = me.Handle;

bool nh;

if (!NeedHeal.TryGetValue(handle, out nh))

NeedHeal.Add(handle, false);

var perc = me.Health/(float) me.MaximumHealth\*100;

//Print(String.Format("Health: {0}, Max Health: {1}, Percent: {2}", Me.Health, Me.MaximumHealth, perc));

if (NeedHeal[handle])

{

if ((perc > 95 &&

me.HasModifiers(new[] {"modifier\_fountain\_aura", "modifier\_fountain\_aura\_buff"}, false)) ||

OrderStates[handle] != OrderState.Escape)

{

NeedHeal[handle] = false;

var newOrder = LastOrderStates[handle] != OrderState.Escape

? LastOrderStates[handle]

: OrderState.Idle;

OrderStates[handle] = \_globalTarget == null ? newOrder/\*OrderState.Idle\*/ : OrderState.InCombo;

//Print(newOrder.ToString());

//Print("im full now ["+handle+"]");

}

else if (!me.HasModifiers(new[] {"modifier\_fountain\_aura", "modifier\_fountain\_aura\_buff"}, false))

{

if (Utils.SleepCheck("move check" + handle))

{

var anyEnemyHero =

Heroes.GetByTeam(me.GetEnemyTeam())

.FirstOrDefault(x => x.IsAlive && x.IsVisible && x.Distance2D(me) <= 800);

if (anyEnemyHero != null)

{

var spell = SpellQ[handle];

if (spell != null && spell.CanBeCasted() && !anyEnemyHero.HasModifier("modifier\_meepo\_earthbind"))

{

spell.CastSkillShot(anyEnemyHero);

}

}

var anyAllyMeepoNearBase =

\_meepoList.Where(

x =>

!Equals(x, me) && x.Distance2D(Fountains.GetAllyFountain()) <= 5000 && x != me &&

!Heroes.GetByTeam(me.GetEnemyTeam()).Any(y => y.IsAlive && y.IsVisible && y.Distance2D(x) <= 1500))

.OrderBy(z => z.Distance2D(Fountains.GetAllyFountain())).FirstOrDefault();

var underTower = Towers.All.Where(x => x.Team == me.GetEnemyTeam())

.Any(x => x.Distance2D(me) <= 800);

if (anyAllyMeepoNearBase != null && w.CanBeCasted() && !underTower)

{

var crossCheck = (me.Distance2D(Fountains.GetAllyFountain()) >=

anyAllyMeepoNearBase.Distance2D(Fountains.GetAllyFountain()))

&& (me.Distance2D(anyAllyMeepoNearBase) >= 500);

if (crossCheck)

{

if (Utils.SleepCheck("poofTimeToBase" + handle))

{

w.UseAbility(anyAllyMeepoNearBase);

Utils.Sleep(2000, "poofTimeToBase" + handle);

}

}

}

var channeledAbility = me.GetChanneledAbility();

var travelBoots = me.FindItem("item\_travel\_boots", true) ??

me.FindItem("item\_travel\_boots\_2", true) ??

me.FindItem("item\_tpscroll", true);

if (me.IsChanneling() && channeledAbility.Name != "item\_travel\_boots"

&& channeledAbility.Name != "item\_travel\_boots\_2" &&

channeledAbility.Name != "item\_tpscroll")

{

//do nothing while in tp

}

else if (!underTower && travelBoots != null && travelBoots.CanBeCasted() &&

me.Distance2D(Fountains.GetAllyFountain()) >= 5000 && Utils.SleepCheck("tp\_cd" + handle) && Utils.SleepCheck("poofTimeToBase" + handle))

{

Utils.Sleep(250, "tp\_cd" + handle);

travelBoots.UseAbility(Fountains.GetAllyFountain().Position);

}

else

{

me.Move(Fountains.GetAllyFountain().Position);

}

Utils.Sleep(500, "move check"+handle);

}

}

}

else

{

if (perc < Menu.Item("Escape.MinRangePercent").GetValue<Slider>().Value || me.Health <= Menu.Item("Escape.MinRange").GetValue<Slider>().Value)

{

//Print("hp too low, go to fountain. Perc: "+perc);

NeedHeal[handle] = true;

LastOrderStates[handle] = OrderStates[handle] == OrderState.Escape

? OrderState.Idle

: OrderStates[handle];

OrderStates[handle] = OrderState.Escape;

}

else

{

//Print("checking for hp");

}

}

return NeedHeal[handle];

}

}

private static void AutoPush(this Hero me)

{

var handle = me.Handle;

var creeps = Creeps.All.Where(x => x != null && x.IsValid && x.IsAlive && x.IsVisible).ToList();

var creepsEnemy = creeps.Where(x => x.Team != MyHero.Team).ToList();

var creepsAlly = creeps.Where(x => x.Team == MyHero.Team).ToList();

var enemyHeroes = Heroes.GetByTeam(MyHero.GetEnemyTeam()).Where(x=>x.IsAlive).ToList();

var towers = Towers.All.Where(x => x.Team != MyHero.Team).Where(x => x.IsAlive).ToList();

var creepWithEnemy =

creepsAlly.FirstOrDefault(

x => x.MaximumHealth\*65/100 < x.Health && creepsEnemy.Any(y => y.Distance2D(x) <= 1000));

var travelBoots = me.FindItem("item\_travel\_boots") ?? me.FindItem("item\_travel\_boots\_2");

if (travelBoots != null && creepWithEnemy != null && Menu.Item("AutoPush.TravelBoots").GetValue<bool>() && Utils.SleepCheck("TravelBoots."+handle))

{

if (travelBoots.CanBeCasted() && !creepsEnemy.Any(x => x.Distance2D(me) <= 1000))

{

travelBoots.UseAbility(creepWithEnemy);

Utils.Sleep(500, "TravelBoots."+handle);

return;

}

}

var nearestTower =

towers.OrderBy(y => y.Distance2D(me))

.FirstOrDefault() ?? Fountains.GetEnemyFountain();

var fountain = Fountains.GetAllyFountain();

var curlane = GetCurrentLane(me);

var clospoint = GetClosestPoint(curlane);

var useThisShit = clospoint.Distance2D(fountain) - 250 > me.Distance2D(fountain);

var name = MeepoSet.Find(x => x.Handle == me.Handle).Handle.ToString();

if (nearestTower != null && Utils.SleepCheck(name + "attack"))

{

var pos = curlane == "mid" || !useThisShit ? nearestTower.Position : clospoint;

var dist = Menu.Item("AutoPush.EscapeRange").GetValue<Slider>().Value;

if (Menu.Item("AutoPush.EscapeFromAnyEnemyHero").GetValue<bool>() &&

enemyHeroes.Any(x => x.Distance2D(me) <= dist)) //escape from hero

{

OrderStates[handle] = OrderState.Escape;

NeedHeal[handle] = true;

}

else if (creepsAlly.Any(x => x.Distance2D(nearestTower) <= 800) ||

me.Distance2D(nearestTower) > 800)

{

//under tower

var hpwasChanged = CheckForChangedHealth(me);

if (hpwasChanged)

{

var allyCreep = creepsAlly.OrderBy(x => x.Distance2D(me)).First();

if (allyCreep != null)

{

var towerPos = nearestTower.Position;

var ang = allyCreep.FindAngleBetween(towerPos, true);

var p = new Vector3((float) (allyCreep.Position.X - 250\*Math.Cos(ang)),

(float) (allyCreep.Position.Y - 250\*Math.Sin(ang)), 0);

me.Move(p);

me.Attack(allyCreep, true);

Utils.Sleep(1200, name + "attack");

}

else

{

var towerPos = nearestTower.Position;

var ang = me.FindAngleBetween(towerPos, true);

var p = new Vector3((float) (towerPos.X - 1250\*Math.Cos(ang)),

(float) (towerPos.Y - 1250\*Math.Sin(ang)), 0);

me.Move(p);

Utils.Sleep(500, name + "attack");

}

}

else

{

var act = me.NetworkActivity;

if (!Utils.SleepCheck("attack\_time" + name))

return;

if (Menu.Item("AutoPush.LastHitMode").GetValue<bool>())

{

var bestEnemyCreep =

creepsEnemy.Where(x => x.Health < me.DamageAverage && x.Distance2D(me) <= 800)

.OrderBy(x => x.Distance2D(me))

.FirstOrDefault();

if (bestEnemyCreep != null)

{

me.Attack(bestEnemyCreep);

Utils.Sleep(UnitDatabase.GetAttackPoint(me)\*1000, "attack\_time" + name);

}

else

{

/\*if (act == NetworkActivity.Attack || act == NetworkActivity.Attack2)

{

me.Stop();

}\*/

if (act == NetworkActivity.Idle)

{

me.Attack(pos);

}

}

}

else

{

if (act == NetworkActivity.Idle) me.Attack(pos);

}

if (Menu.Item("AutoPush.AutoW").GetValue<bool>() && SpellW[handle] != null)

{

var w = SpellW[handle];

var castRange = w.GetRealCastRange();

if (w.CanBeCasted() &&

creepsEnemy.Any(x => x.Distance2D(me) <= castRange && x.Health <= 60 + 20\*w.Level) &&

Utils.SleepCheck("w\_push" + name))

{

w.UseAbility(me);

Utils.Sleep(250, "w\_push" + name);

}

}

}

Utils.Sleep(100, name + "attack");

}

else

{

var towerPos = nearestTower.Position;

var ang = me.FindAngleBetween(towerPos, true);

var p = new Vector3((float) (me.Position.X - 1000\*Math.Cos(ang)),

(float) (me.Position.Y - 1000\*Math.Sin(ang)), 0);

me.Move(p);

Utils.Sleep(200, name + "attack");

}

}

}

private static void Stacking(this Hero me)

{

var s = JungleCamps.FindClosestCampForStacking(me, Menu.Item("JungleStack.TeamCheck").GetValue<bool>(),

Menu.Item("JungleStack.Ancient").GetValue<bool>());

var enemyHeroes = Heroes.GetByTeam(me.GetEnemyTeam()).Where(x => x.IsAlive).ToList();

var dist = Menu.Item("JungleStack.EscapeRange").GetValue<Slider>().Value;

if (Menu.Item("JungleStack.EscapeFromAnyEnemyHero").GetValue<bool>() &&

enemyHeroes.Any(x => x.Distance2D(me) <= dist)) //escape from hero

{

var handle = me.Handle;

OrderStates[handle] = OrderState.Escape;

NeedHeal[handle] = true;

}

if (s == null) return;

s.Stacking = me;

var set = MeepoSet.Find(x => Equals(x.Hero, me));

var name = set.Handle.ToString();

var sec = Game.GameTime%60;

var timeForStart = s.WaitPosition.Distance2D(s.CampPosition)/me.MovementSpeed;

var time = s.StackTime - timeForStart - sec;

//Print("Current Time: [" + sec + "] Time For Travel: [" + timeForStart + "] TimeForStartMoving: [" + (time - sec) + "]");

//Print(time.ToString());

var min = Math.Floor(Game.GameTime/60)%2 == 0;

if (time >= 0.5)

{

if (Utils.SleepCheck("move\_cd2" + name))

{

me.Move(s.WaitPosition);

Utils.Sleep(250, "move\_cd2" + name);

}

}

else if (Utils.SleepCheck("move\_cd" + name) && min)

{

var pos = s.CampPosition;

var ang = me.FindAngleBetween(pos, true);

var p = new Vector3((float) (pos.X - 80\*Math.Cos(ang)),

(float) (pos.Y - 80\*Math.Sin(ang)), 0);

me.Move(p);

me.Move(s.StackPosition, true);

Utils.Sleep((60 - s.StackTime)\*1000 + 8000, "move\_cd" + name);

Log.Debug($"Meepo #[{MeepoSet.Find(settings => settings.Hero.Equals(me))?.Id}] trying to stack camp #[{s.Id}] ({s.Name})");

// TODO: camp 4 & 3

}

}

#endregion

#region OtherShit

private static void DrawEffects(Meepo meepo, Hero target)

{

ParticleEffect effect;

var handle = meepo.Handle;

if (!Effects.TryGetValue(handle, out effect))

{

Effects.Add(handle, new ParticleEffect(@"particles\ui\_mouseactions\range\_finder\_tower\_aoe.vpcf", target));

}

if (effect == null) return;

effect.SetControlPoint(2, new Vector3(meepo.Position.X, meepo.Position.Y, meepo.Position.Z));

effect.SetControlPoint(6, new Vector3(1, 0, 0));

effect.SetControlPoint(7, new Vector3(target.Position.X, target.Position.Y, target.Position.Z));

}

private static void FlushEffect()

{

foreach (var meepo in \_meepoList)

{

ParticleEffect effect;

var handle = meepo.Handle;

Ability w;

if (!SpellW.TryGetValue(handle, out w))

SpellW[handle] = meepo.Spellbook.Spell2;

switch (OrderStates[handle])

{

case OrderState.Laning:

AutoPush(meepo);

break;

case OrderState.Jungle:

JungleFarm(meepo);

break;

case OrderState.Stacking:

Stacking(meepo);

break;

}

if (OrderStates[handle] == OrderState.InCombo) OrderStates[handle] = OrderState.Idle;

if (!Effects.TryGetValue(handle, out effect)) continue;

effect.Dispose();

Effects.Remove(handle);

}

}

private static bool NeedUseMeka()

{

return \_meepoList.Where(x => x.IsAlive && x.Distance2D(MyHero) <= 900).Any(meepo => meepo.Health <= 500);

}

private static readonly Dictionary<uint, bool> NeedHeal = new Dictionary<uint, bool>();

private static bool \_leftMouseIsPress;

private static readonly Dictionary<Unit, uint> LastCheckedHp = new Dictionary<Unit, uint>();

private static bool CheckForChangedHealth(Unit me)

{

uint health;

if (!LastCheckedHp.TryGetValue(me, out health))

{

LastCheckedHp.Add(me, me.Health);

}

var boolka = health > me.Health;

LastCheckedHp[me] = me.Health;

return boolka;

}

private static string GetCurrentLane(Unit me)

{

return LaneDictionary.OrderBy(x => x.Key.Distance2D(me)).First().Value;

}

private static Vector3 GetClosestPoint(string pos)

{

var list = LaneDictionary.Keys.ToList();

switch (pos)

{

case "top":

return list[0];

case "bot":

return list[3];

default:

return list[6];

}

}

public static readonly List<MeepoSettings> MeepoSet = new List<MeepoSettings>();

private static void RefreshMeepoList()

{

if (Utils.SleepCheck("SelectChecker"))

{

\_selectedMeepo = MyPlayer.Selection.Where(x => x.ClassId == ClassId.CDOTA\_Unit\_Hero\_Meepo).ToList();

//Print("selected count: " + SelectedMeepo.Count);

Utils.Sleep(150, "SelectChecker");

}

if (!Utils.SleepCheck("MeepoRefresh")) return;

Utils.Sleep(500, "MeepoRefresh");

if (\_meepoList.Count >= 1 + \_ultimate.Level + (MyHero.AghanimState() ? 1 : 0)) return;

\_meepoList =

ObjectManager.GetEntities<Meepo>()

.Where(x => x.IsValid && !x.IsIllusion() && x.Team == MyHero.Team) /\*.OrderBy(x => x.Handle)\*/

.ToList();

Print("meepo count is " + \_meepoList.Count + " AGHANIM STATE: " + MyHero.AghanimState());

foreach (var meepo in \_meepoList)

{

var handle = meepo.Handle;

OrderState state;

OrderState laststate;

if (!OrderStates.TryGetValue(handle, out state))

{

OrderStates.Add(handle, OrderState.Idle);

}

if (!LastOrderStates.TryGetValue(handle, out laststate))

{

LastOrderStates.Add(handle, OrderState.Idle);

}

Ability q, w;

if (!SpellQ.TryGetValue(handle, out q))

SpellQ[handle] = meepo.Spellbook.Spell1;

if (!SpellW.TryGetValue(handle, out w))

SpellW[handle] = meepo.Spellbook.Spell2;

}

foreach (var meepo in \_meepoList.Where(meepo => !MeepoSet.Any(x => Equals(x.Hero, meepo))))

{

MeepoSet.Add(new MeepoSettings(meepo));

}

}

private static void Camp\_update(EventArgs args)

{

if (!Utils.SleepCheck("Camp.Update"))

return;

Utils.Sleep(150, "Camp.Update");

//var refreshTime = (Game.GameTime + 60)%120 == 0;

foreach (var camp in JungleCamps.GetCamps)

{

foreach (

var enemy in

from meepo in

MeepoSet.Where(x => x.CurrentOrderState == OrderState.Jungle && x.IsAlive)

.Select(y => y.Hero)

let heroDist = meepo.Distance2D(camp.CampPosition)

where heroDist < 100

select ObjectManager.GetEntities<Unit>()

.Any(

x =>

x.IsAlive && x.IsVisible && x.Team != MyHero.Team && x.Distance2D(meepo) <= 500 &&

!x.IsWaitingToSpawn))

{

camp.CanBeHere = enemy;

if (enemy || camp.Delayed) continue;

camp.Delayed = true;

//Print("[CampStatus]: add delay" + (120 - (Game.GameTime + 60) % 120),true);

DelayAction.Add((120 - (Game.GameTime + 60) % 120)\*1000, () =>

{

camp.Delayed = false;

camp.CanBeHere = true;

//Print("[CampStatus]: delayAction" + camp.CanBeHere,true);

});

}

}

}

private static Hero ClosestToMouse(Hero source, float range = 600)

{

var mousePosition = Game.MousePosition;

var enemyHeroes = ObjectManager.GetEntities<Hero>()

.Where(

x =>

x.Team == source.GetEnemyTeam() && !x.IsIllusion && x.IsAlive && x.IsVisible &&

x.Distance2D(mousePosition) <= range /\*&& !x.IsMagicImmune()\*/)

.OrderBy(x => x.Distance2D(mousePosition));

return enemyHeroes.FirstOrDefault();

}

private static void Print(string s,bool print=true)

{

if (print)

Game.PrintMessage(s);

}

private static string PrintVector(this Vector3 s)

{

return $"({s.X}:{s.Y}:{s.Z})";

}

private static float GetRealCastRange(this Ability ability)

{

var range = ability.CastRange;

if (range >= 1) return range;

var data =

ability.AbilitySpecialData.FirstOrDefault(

x => x.Name.Contains("radius") || (x.Name.Contains("range") && !x.Name.Contains("ranged")));

if (data == null) return range;

var level = ability.Level == 0 ? 0 : ability.Level - 1;

range = (uint) (data.Count > 1 ? data.GetValue(level) : data.Value);

return range;

}

private static void DrawText(Vector2 a, float w, float h, Color @on, string drawOnButtonText = "")

{

var textSize = Drawing.MeasureText(

drawOnButtonText,

"Arial",

new Vector2((float) (h\*.50), 100),

FontFlags.AntiAlias);

var textPos = a + new Vector2(5, (float) ((h\*0.5) - (textSize.Y\*0.5)));

Drawing.DrawRect(a, new Vector2(w, h), @on);

Drawing.DrawText(

drawOnButtonText,

textPos,

new Vector2((float) (h\*.50), 100),

Color.White,

FontFlags.AntiAlias | FontFlags.Additive | FontFlags.Custom);

}

private static void DrawButton(Vector2 a, float w, float h, ref bool clicked, Color @on, Color off,

string drawOnButtonText = "", bool isSelected = false)

{

var isIn = Utils.IsUnderRectangle(Game.MouseScreenPosition, a.X, a.Y, w, h);

if (\_leftMouseIsPress && Utils.SleepCheck("ClickButtonCd") && isIn)

{

clicked = !clicked;

Utils.Sleep(250, "ClickButtonCd");

}

var newColor = isIn

? new Color((int) (clicked ? @on.R : off.R), clicked ? @on.G : off.G, clicked ? @on.B : off.B, 150)

: clicked ? @on : off;

var textSize = Drawing.MeasureText(

drawOnButtonText,

"Arial",

new Vector2((float) (h\*.50), 100),

FontFlags.AntiAlias);

var textPos = a + new Vector2(5, (float) ((h\*0.5) - (textSize.Y\*0.5)));

Drawing.DrawRect(a, new Vector2(w, h), newColor);

Drawing.DrawText(

drawOnButtonText,

textPos,

new Vector2((float) (h\*.50), 100),

Color.White,

FontFlags.AntiAlias | FontFlags.Additive | FontFlags.Custom);

if (isSelected)

{

Drawing.DrawRect(a, new Vector2(w, h), Color.YellowGreen, true);

}

}

private static void DrawButton(Vector2 a, float w, float h, ref int id, int needed, Color @on,

string drawOnButtonText = "")

{

var isIn = Utils.IsUnderRectangle(Game.MouseScreenPosition, a.X, a.Y, w, h);

if (\_leftMouseIsPress && Utils.SleepCheck("ClickButtonCd") && isIn)

{

id = needed;

Utils.Sleep(250, "ClickButtonCd");

}

var newColor = @on;

var textSize = Drawing.MeasureText(

drawOnButtonText,

"Arial",

new Vector2((float) (h\*.50), 100),

FontFlags.AntiAlias);

var textPos = a + new Vector2(5, (float) ((h\*0.5) - (textSize.Y\*0.5)));

Drawing.DrawRect(a, new Vector2(w, h), newColor);

**项目后1500行**

private static readonly float IconSize;

public static Vector2 ScreenSize { get; set; }

static HeroPickStageScreenHelper()

{

float size;

float rightCoef;

float leftCoef;

ScreenSize = new Vector2(Drawing.Width, Drawing.Height);

var ratio = Math.Floor((decimal)(ScreenSize.X / ScreenSize.Y \* 100));

switch ((int)ratio)

{

case 160: //16:10

leftCoef = 2.45f;

rightCoef = 1.68f;

size =14f;

break;

case 125: //4:3

leftCoef = 2.46f;

rightCoef = 1.68f;

size =14.3f;

break;

case 177: //16:9

leftCoef = 2.47f;

rightCoef = 1.68f;

size = 15.6f;

break;

default:

Printer.PrintError(

@"Your screen resolution is not supported and drawings might have wrong size/position, (" +

ratio + ")");

leftCoef = 2.47f;

rightCoef = 1.68f;

size = 15.6f;

break;

}

FirstLeftHeroPos = ScreenSize.X / leftCoef;

FirstRightHeroPos = ScreenSize.X / rightCoef;

IconSize = ScreenSize.X/size;

//Console.WriteLine($"Rate: {ratio} Left: {FirstLeftHeroPos} Right: {FirstRightHeroPos} size: {IconSize} Center: {centerOfScreen}");

}

public static float GetPlayerPosition(int id)

{

float pos;

if (id > 4)

{

//var extra = id > 4 ? IconSize\*(id - 4) : 0;

var extra = IconSize\*(id - 5);

pos = FirstRightHeroPos + extra;

}

else

pos = FirstLeftHeroPos - IconSize\*(5-id);

return pos;

}

}

public class PlayerInfo

{

public string Matches { get; set; }

public string Name { get; set; }

public int Id;

public int Solo;

public int Party;

public string Country;

public string PossibleMmr;

public string Wr;

public Hero Hero;

public string TotalGames;

public string Wins;

public int WrOnCurrentHero;

public PlayerInfo(int id, int solo, int party, string country, string possibleMmr, string winrate, string matches, string name)

{

Matches = matches;

Id = id;

Solo = solo;

Party = party;

Country = country;

PossibleMmr = possibleMmr;

Wr = winrate;

Name = name.Length > 10 ? name.Substring(0, 10) : name;

}

public PlayerInfo(int id, int solo, int party, string country, string possibleMmr, string winrate,

string matches, string name, Hero hero, string totalGames, string wins, int wrOnCurrentHero)

: this(id, solo, party, country, possibleMmr, winrate, matches, name)

{

Hero = hero;

TotalGames = totalGames;

Wins = wins;

WrOnCurrentHero = wrOnCurrentHero;

}

}

public class OpenDota

{

private static readonly Font Text;

private static readonly ILog Log = AssemblyLogs.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

private static readonly List<PlayerInfo> PlayerInfoList;

private static bool \_loaded;

private static readonly bool Init;

private static bool Check => Game.GameState == GameState.PreGame;

private static bool IsEnable => Members.Menu.Item("OpenDota.Enable").GetValue<bool>();

static OpenDota()

{

if (!IsEnable)

return;

if (Init)

return;

Init = true;

Printer.PrintInfo("[OpenDota] Init");

/\*if (Drawing.RenderMode != RenderMode.Dx9)

{

Printer.Print("You're using not dx9, OpenDota helper will not working!", true);

Printer.PrintError("You're using not dx9, OpenDota helper will not working!");

return;

}\*/

/\*Console.WriteLine(

$"Screen Size: {HeroPickStageScreenHelper.ScreenSize.X}/{HeroPickStageScreenHelper.ScreenSize.Y}");\*/

//SingleFake();

PlayerInfoList = new List<PlayerInfo>();

/\*if (Drawing.Direct3DDevice9 != null)

Text = new Font(

Drawing.Direct3DDevice9,

new FontDescription

{

FaceName = "Tahoma",

Height = 14,

OutputPrecision = FontPrecision.Default,

Quality = FontQuality.Default

});\*/

Game.OnUpdate += GameOnOnUpdate;

/\*AppDomain.CurrentDomain.DomainUnload += CurrentDomainDomainUnload;

Drawing.OnPostReset += Drawing\_OnPostReset;

Drawing.OnPreReset += Drawing\_OnPreReset;

Drawing.OnEndScene += Drawing\_OnEndScene;\*/

Drawing.OnDraw+=DrawingOnOnDraw;

Events.OnLoad += (sender, args) =>

{

//DelayAction.Add(1000, SingleFake);

};

//DelayAction.Add(1000,PartyFake);

}

private static void DrawingOnOnDraw(EventArgs args)

{

try

{

if (!Checker.IsActive()) return;

if (!Members.Menu.Item("OpenDota.Enable").GetValue<bool>()) return;

}

catch (Exception)

{

// ignored

}

if (Check)

{

var newlist = PlayerInfoList.ToList();

foreach (var playerInfo in newlist)

{

var id = playerInfo.Id;

var startXPosition = HeroPickStageScreenHelper.GetPlayerPosition(id);

var position = new Vector2(startXPosition, 35);

var size = HudInfoNew.GetTopPanelSizeY();

position += new Vector2(0, (float)size \* 1.8f);

var defClr = Color.White;

//DrawShadowText(playerInfo.Name, (int)position.X, (int)position.Y, defClr);

Drawing.DrawText($"[{playerInfo.Name}]", position, defClr, FontFlags.None);

position.Y += 15;

//DrawShadowText(playerInfo.Wr, (int)position.X, (int)position.Y, defClr);

Drawing.DrawText($"[{playerInfo.Wr}]", position, defClr, FontFlags.None);

position.Y += 15;

/\*DrawShadowText(

playerInfo.Solo == 0 ? $"Estimated: {playerInfo.PossibleMmr}" : $"Solo: {playerInfo.Solo}",

(int)position.X, (int)position.Y, defClr);\*/

Drawing.DrawText(

playerInfo.Solo == 0 ? $"Estimated: {playerInfo.PossibleMmr}" : $"Solo: {playerInfo.Solo}",

position, defClr, FontFlags.None);

if (playerInfo.Party > 0)

{

position.Y += 15;

//DrawShadowText($"Party: {playerInfo.Party}", (int)position.X, (int)position.Y, defClr);

Drawing.DrawText($"[{playerInfo.Party}]", position, defClr, FontFlags.None);

}

var gameHistorySize = playerInfo.Matches.Length - 2;

if (gameHistorySize >= 1)

{

position.Y += 15;

for (var i = 0; i < gameHistorySize; i++)

{

var isTrue = playerInfo.Matches[i + 1] == '+';

var clr = isTrue ? Color.Green : Color.Red;

position.X += 10;

var text = '⬤';//●

//DrawShadowText($"{text}", (int)position.X, (int)position.Y, clr);

Drawing.DrawText($"[{text}]", position, clr, FontFlags.None);

}

}

if (playerInfo.Country.Length > 0)

{

try

{

var n = Convert.ToInt32(playerInfo.Country);

if (n > 0)

{

position.Y += 15;

//DrawShadowText($"[{playerInfo.Country}]", (int) position.X, (int) position.Y, defClr);

Drawing.DrawText($"[{playerInfo.Country}]", position, defClr, FontFlags.None);

}

}

catch (Exception)

{

}

}

if (playerInfo.TotalGames.Length > 0)

{

try

{

var n = Convert.ToInt32(playerInfo.TotalGames);

if (n == 0)

continue;

}

catch (Exception)

{

continue;

}

var totalGames = Convert.ToInt32(playerInfo.TotalGames);

var wins = Convert.ToInt32(playerInfo.Wins);

var loses = totalGames - wins;

var wr = playerInfo.WrOnCurrentHero;

position.Y += 15;

position.X = startXPosition;

Drawing.DrawText($"[{playerInfo.Hero?.GetRealName()}: {wins}/{loses} ({wr}%)]", position, defClr,

FontFlags.None);

//DrawShadowText($"[{playerInfo.Hero?.GetRealName()}: {wins}/{loses} ({wr}%)]", (int)position.X, (int)position.Y, defClr);

}

}

}

}

private static void GameOnOnUpdate(EventArgs args)

{

try

{

if (!Checker.IsActive()) return;

if (!Members.Menu.Item("OpenDota.Enable").GetValue<bool>()) return;

}

catch (Exception)

{

// ignored

}

if (\_loaded)

return;

if (Check)

{

\_loaded = true;

Console.WriteLine("[OpenDota] [pre]Loaded");

DelayAction.Add(500, () =>

{

Console.WriteLine("[OpenDota] Loaded");

Beeeeaaaaaar();

});

//Beeeeaaaaaar();

}

}

private static async void Beeeeaaaaaar()

{

for (uint i = 0; i < Game.MaximumClients; i++)

{

var player = ObjectManager.GetPlayerById(i);

if (player == null || !player.IsValid || player.IsFakeClient)

continue;

try

{

Printer.PrintSuccess(new string('-', Console.BufferWidth));

var steamId = player.PlayerSteamId;

Log.Debug($"Player({i}): {player.Name} => id: {steamId}");

if (steamId <= 10)

{

Log.Error("Wrong steam id!");

continue;

}

var test = await FindWinRateAsync(steamId);

if (test < 0 || test > 100)

{

Log.Error("Cant load this player!");

continue;

}

var playerReq = await GetPlayerAsync(steamId);

var wr = await FindFullWinRateAsync(steamId);

//var accName = GetValue("personaname\":", playerReq);

int estimate = 0;

int stdDev = 0;

int solo = 0;

int party = 0;

string country = "";

string possibleMmr = "";

string matches = "";

string infoAboutHero = "";

/\*int estimate = Convert.ToInt32(GetValue("estimate\":", playerReq));

int stdDev = Convert.ToInt32(GetValue("stdDev\":", playerReq));

int solo = Convert.ToInt32(GetValue("solo\_competitive\_rank\":", playerReq));

int party = Convert.ToInt32(GetValue("competitive\_rank\":", playerReq));

var country = GetValue("loccountrycode\":", playerReq);

var possibleMmr = $"{estimate - stdDev}-{estimate + stdDev}";\*/

try

{

//Console.WriteLine("estimate: "+ GetValue("estimate\":", playerReq));

estimate = Convert.ToInt32(GetValue("{\"estimate\":", playerReq));

}

catch (Exception)

{

//Log.Error("1");

}

try

{

var item = GetValue("stdDev\":", playerReq);

item = item.Substring(0, item.IndexOf(".", StringComparison.Ordinal));

stdDev = Convert.ToInt32(item);

}

catch (Exception)

{

//Log.Error("2");

}

try

{

var item = GetValue("solo\_competitive\_rank\":", playerReq);

solo = Convert.ToInt32(item/\*.Substring(1, item.Length - 2)\*/);

}

catch (Exception)

{

//Log.Error("3");

}

try

{

var item = GetValue("\"competitive\_rank\":", playerReq);

party = Convert.ToInt32(item/\*.Substring(1, item.Length - 2)\*/);

}

catch (Exception)

{

//Log.Error("4");

}

try

{

country = GetValue("loccountrycode\":", playerReq);

}

catch (Exception)

{

//Log.Error("5");

}

try

{

possibleMmr = $"{estimate - stdDev}-{estimate + stdDev}";

}

catch (Exception)

{

//Log.Error("6");

}

try

{

matches = await FindMatches(steamId);

}

catch (Exception)

{

//Log.Error("7");

}

try

{

infoAboutHero = await FindInfoAboutHero(steamId, (uint)player.Hero.HeroId);

}

catch (Exception)

{

//Log.Error("8");

}

//Log.Debug("test: "+ matches);

Log.Debug(

$"[WinRate: {wr}] [solo: {solo}] [party {party}] [estimate mmr: {possibleMmr}] [{country}] history: {matches}");

string totalGames = "";

string wins = "";

try

{

totalGames = GetValue("games", infoAboutHero).TrimStart(':');

}

catch (Exception)

{

}

try

{

wins = GetValue("win", infoAboutHero).TrimStart(':');

}

catch (Exception)

{

}

var wrOnCurrentHero = 0;

try

{

wrOnCurrentHero = (int)((float)Convert.ToInt32(wins) / Convert.ToInt32(totalGames) \* 100.0f);

}

catch (Exception)

{

}

try

{

Log.Debug(

$"[Hero: {player?.Hero?.GetRealName()} -> [Games {totalGames}] [Wins {wins}] [WR {wrOnCurrentHero}%]");

}

catch (Exception)

{

}

PlayerInfoList.Add(new PlayerInfo((int)i, solo, party, country, possibleMmr, wr, matches, player?.Name, player?.Hero, totalGames, wins, wrOnCurrentHero));

}

catch (Exception e)

{

Log.Debug($"error with player: {player.Name} ({i}) -> {e}");

}

}

}

private static void Drawing\_OnEndScene(EventArgs args)

{

try

{

if (!Checker.IsActive()) return;

if (!Members.Menu.Item("OpenDota.Enable").GetValue<bool>()) return;

}

catch (Exception)

{

// ignored

}

if (Drawing.Direct3DDevice9 == null || Drawing.Direct3DDevice9.IsDisposed)

{

return;

}

if (Check)

{

var newlist = PlayerInfoList.ToList();

foreach (var playerInfo in newlist)

{

var id = playerInfo.Id;

var startXPosition = HeroPickStageScreenHelper.GetPlayerPosition(id);

var position = new Vector2(startXPosition, 35);

var size = HudInfoNew.GetTopPanelSizeY();

position += new Vector2(0, (float)size \* 1.8f);

var defClr = Color.White;

DrawShadowText(playerInfo.Name, (int)position.X, (int)position.Y, defClr);

position.Y += 15;

DrawShadowText(playerInfo.Wr, (int)position.X, (int)position.Y, defClr);

position.Y += 15;

DrawShadowText(

playerInfo.Solo == 0 ? $"Estimated: {playerInfo.PossibleMmr}" : $"Solo: {playerInfo.Solo}",

(int)position.X, (int)position.Y, defClr);

if (playerInfo.Party > 0)

{

position.Y += 15;

DrawShadowText($"Party: {playerInfo.Party}", (int)position.X, (int)position.Y, defClr);

}

var gameHistorySize = playerInfo.Matches.Length - 2;

if (gameHistorySize >= 1)

{

position.Y += 15;

for (var i = 0; i < gameHistorySize; i++)

{

var isTrue = playerInfo.Matches[i + 1] == '+';

var clr = isTrue ? Color.Green : Color.Red;

position.X += 10;

var text = '⬤';//●

DrawShadowText($"{text}", (int)position.X, (int)position.Y, clr);

}

}

if (playerInfo.Country.Length > 0)

{

try

{

var n = Convert.ToInt32(playerInfo.Country);

if (n > 0)

{

position.Y += 15;

DrawShadowText($"[{playerInfo.Country}]", (int) position.X, (int) position.Y, defClr);

}

}

catch (Exception)

{

}

}

if (playerInfo.TotalGames.Length > 0)

{

try

{

var n = Convert.ToInt32(playerInfo.TotalGames);

if (n == 0)

continue;

}

catch (Exception)

{

continue;

}

var totalGames = Convert.ToInt32(playerInfo.TotalGames);

var wins = Convert.ToInt32(playerInfo.Wins);

var loses = totalGames-wins;

var wr = playerInfo.WrOnCurrentHero;

position.Y += 15;

position.X = startXPosition;

DrawShadowText($"[{playerInfo.Hero?.GetRealName()}: {wins}/{loses} ({wr}%)]", (int)position.X, (int)position.Y, defClr);

}

}

}

}

#region Fakes

private static async void PartyFake()

{

Log.Debug("starting fake");

if (true)

{

\_loaded = true;

for (uint i = 0; i < 10; i++)

{

try

{

Printer.PrintSuccess(new string('-', Console.BufferWidth));

uint steamId = 105248644;

Log.Debug($"Player({i}): {"FAKE"} => id: {steamId}");

if (steamId <= 10)

{

Log.Error("Wrong steam id!");

continue;

}

var test = await FindWinRateAsync(steamId);

if (test < 0 || test > 100)

{

Log.Error("Cant load this player!");

continue;

}

var playerReq = await GetPlayerAsync(steamId);

var wr = await FindFullWinRateAsync(steamId);

//var accName = GetValue("personaname\":", playerReq);

int estimate = 0;

int stdDev = 0;

int solo = 0;

int party = 0;

string country = "";

string possibleMmr = "";

string matches = "";

/\*int estimate = Convert.ToInt32(GetValue("estimate\":", playerReq));

int stdDev = Convert.ToInt32(GetValue("stdDev\":", playerReq));

int solo = Convert.ToInt32(GetValue("solo\_competitive\_rank\":", playerReq));

int party = Convert.ToInt32(GetValue("competitive\_rank\":", playerReq));

var country = GetValue("loccountrycode\":", playerReq);

var possibleMmr = $"{estimate - stdDev}-{estimate + stdDev}";\*/

try

{

//Console.WriteLine("estimate: "+ GetValue("estimate\":", playerReq));

estimate = Convert.ToInt32(GetValue("{\"estimate\":", playerReq));

}

catch (Exception)

{

//Log.Error("1");

}

try

{

var item = GetValue("stdDev\":", playerReq);

item = item.Substring(0, item.IndexOf(".", StringComparison.Ordinal));

stdDev = Convert.ToInt32(item);

}

catch (Exception)

{

//Log.Error("2");

}

try

{

var item = GetValue("solo\_competitive\_rank\":", playerReq);

solo = Convert.ToInt32(item /\*.Substring(1, item.Length - 2)\*/);

}

catch (Exception)

{

//Log.Error("3");

}

try

{

var item = GetValue("\"competitive\_rank\":", playerReq);

party = Convert.ToInt32(item /\*.Substring(1, item.Length - 2)\*/);

}

catch (Exception)

{

//Log.Error("4");

}

try

{

country = GetValue("loccountrycode\":", playerReq);

}

catch (Exception)

{

//Log.Error("5");

}

try

{

possibleMmr = $"{estimate - stdDev}-{estimate + stdDev}";

}

catch (Exception)

{

//Log.Error("6");

}

try

{

matches = await FindMatches(steamId);

}

catch (Exception)

{

//Log.Error("7");

}

//Log.Debug("test: "+ matches);

PlayerInfoList.Add(new PlayerInfo((int)i, solo, party, country, possibleMmr, wr, matches,

"FAKE "+i));

Log.Debug(

$"[WinRate: {wr}] [solo: {solo}] [party {party}] [estimate mmr: {possibleMmr}] [{country}] history: {matches}");

//var success = await TryToFindPlayerAsync(player.Name);

//var success = TryToFindPlayer(player.Name);

/\*Log.Debug($"Try To Find player with search API -> player({i}): " + player.Name + " -> (steamId)" +

player.PlayerSteamID +

$" -> success: {success.Length != 2} bufferSize [{success.Length}]");

if (success.Length != 2)

{

var accId = GetValue("account\_id\":", success);

var accName = GetValue("personaname\":", success);

Log.Debug("id: " + accId);

Log.Debug("personaname: " + accName);

Log.Debug("wr: " + FindWinRate(Convert.ToUInt32(accId)) + "%");

//Log.Debug("GetPlayer: " + GetPlayer(accId));

}

else

{

Log.Debug($"cant find {player.Name}!");

}\*/

}

catch (Exception e)

{

Log.Debug($"error with player: ({i}) -> {e}");

}

}

}

Log.Debug("ending fake");

}

private static async void SingleFake()

{

Log.Debug("loading!");

var s = await GetPlayerAsync(1);

Log.Debug(s);

s = await FindInfoAboutHero(1, (uint) ObjectManager.LocalHero.HeroId);

Log.Debug(s);

var totalGames = GetValue("games", s).TrimStart(':');

var wins = GetValue("win", s).TrimStart(':');

var wrOnCurrentHero = (float)Convert.ToInt32(wins) / Convert.ToInt32(totalGames) \* 100.0f;

Log.Debug(

$"[Hero: {ObjectManager.LocalHero.GetRealName()} -> [Games {totalGames}] [Wins {wins}] [WR {wrOnCurrentHero}%]");

}

#endregion

#region render stuff

private static Vector2 DrawText(string text, Vector2 tSize, Vector2 startPos)

{

var textSize = Drawing.MeasureText(text, "Arial",

tSize, FontFlags.None);

Drawing.DrawRect(startPos, textSize, new Color(0, 0, 0, 155));

var textPos = startPos;

Drawing.DrawText(

text,

textPos, tSize,

Color.White,

FontFlags.AntiAlias | FontFlags.StrikeOut);

return textSize;

}

private static Vector2 DrawHeroIcon(Hero target, Vector2 size, Vector2 startPos)

{

var extra = new Vector2(size.X / 3, 0);

var finalSize = size + extra;

Drawing.DrawRect(startPos, finalSize, Textures.GetHeroTexture(target.StoredName()));

Drawing.DrawRect(startPos, finalSize, new Color(0, 0, 0, 255), true);

return finalSize;

}

private static void DrawShadowText(string stext, int x, int y, Color clr)

{

Text.DrawText(null, stext, x + 1, y + 1, Color.Black);

Text.DrawText(null, stext, x, y, clr);

}

private static void CurrentDomainDomainUnload(object sender, EventArgs e)

{

try { if (!Checker.IsActive()) return; } catch { }

Text?.Dispose();

}

private static void Drawing\_OnPostReset(EventArgs args)

{

try { if (!Checker.IsActive()) return; } catch { }

Text?.OnLostDevice();

}

private static void Drawing\_OnPreReset(EventArgs args)

{

try { if (!Checker.IsActive()) return; } catch { }

Text?.OnLostDevice();

}

#endregion

#region Helpers

private static string TryToFindPlayer(string name, bool print = false)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/search?q={name}&similarity=1");

string strContent;

using (var response = webRequest.GetResponse())

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

if (print)

Console.WriteLine(strContent);

}

return strContent;

}

private static async Task<string> TryToFindPlayerAsync(string name, bool print = false)

{

var request = WebRequest.Create($"https://api.opendota.com/api/search?q={name}&similarity=1");

string strContent;

using (var response = (HttpWebResponse)await Task.Factory

.FromAsync(request.BeginGetResponse,

request.EndGetResponse,

null))

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

if (print)

Console.WriteLine(strContent);

}

return strContent;

}

private static string GetPlayer(string id)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{id}");

string strContent;

using (var response = webRequest.GetResponse())

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

//Console.WriteLine(strContent);

}

return strContent;

}

private static async Task<string> GetPlayerAsync(uint id)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{id}");

string strContent;

var response = (HttpWebResponse)await Task.Factory

.FromAsync(webRequest.BeginGetResponse,

webRequest.EndGetResponse,

null);

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

//Console.WriteLine(strContent);

}

return strContent;

}

private static int FindWinRate(uint id)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{id}/wl");

//((HttpWebRequest)webRequest).UserAgent = ".NET Framework";

string strContent;

using (var response = webRequest.GetResponse())

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

//Console.WriteLine(strContent);

}

int win = Convert.ToInt32(GetValue("win\":", strContent));

int lose = Convert.ToInt32(GetValue("lose\":", strContent));

return (int)(win / (win + (double)lose) \* 100f);

}

private static async Task<int> FindWinRateAsync(uint id)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{id}/wl");

//((HttpWebRequest)webRequest).UserAgent = ".NET Framework";

string strContent;

var response = (HttpWebResponse)await Task.Factory

.FromAsync(webRequest.BeginGetResponse,

webRequest.EndGetResponse,

null);

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

//Console.WriteLine(strContent);

}

int win = Convert.ToInt32(GetValue("win\":", strContent));

int lose = Convert.ToInt32(GetValue("lose\":", strContent));

return (int)(win / (win + (double)lose) \* 100f);

}

private static async Task<string> FindFullWinRateAsync(uint id)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{id}/wl");

//((HttpWebRequest)webRequest).UserAgent = ".NET Framework";

string strContent;

var response = (HttpWebResponse)await Task.Factory

.FromAsync(webRequest.BeginGetResponse,

webRequest.EndGetResponse,

null);

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

}

int win = Convert.ToInt32(GetValue("win\":", strContent));

int lose = Convert.ToInt32(GetValue("lose\":", strContent));

var wr = (int)(win / (win + (double)lose) \* 100f);

return $"({win}/{lose}) {wr}%";

}

private static async Task<string> FindMatches(uint id, int gameLimit = 5, int gameMode = 22)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{id}/matches?limit={gameLimit}&game\_mode={gameMode}");

//((HttpWebRequest)webRequest).UserAgent = ".NET Framework";

var response = (HttpWebResponse)await Task.Factory

.FromAsync(webRequest.BeginGetResponse,

webRequest.EndGetResponse,

null);

var info = "[";

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

var strContent = reader.ReadToEnd();

for (var i = 0; i < gameLimit; i++)

{

var start = strContent.IndexOf("{", StringComparison.Ordinal);

var end = strContent.IndexOf("}", StringComparison.Ordinal);

if (start == -1 || end == -1)

continue;

var tempLine = strContent.Substring(start, end);

strContent = strContent.Remove(start, end);

//Console.WriteLine(new string('-', Console.BufferWidth));

//Console.WriteLine(tempLine);

var playerSlot = Convert.ToInt32(GetValue("player\_slot\":", tempLine));

var radiantWin = GetValue("radiant\_win\":", tempLine) == "true";

var win = (playerSlot <= 10 && radiantWin) || (playerSlot > 10 && !radiantWin);

info += win ? "+" : "-";

}

}

info += "]";

return info;

}

private static async Task<string> FindInfoAboutHero(uint playerid,uint heroId)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{playerid}/heroes?hero\_id={heroId}") string strContent;

var response = (HttpWebResponse)await Task.Factory

.FromAsync(webRequest.BeginGetResponse,

webRequest.EndGetResponse,

null);

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

//Console.WriteLine(strContent);

}

string trimmed = strContent.Trim();

var end = trimmed.Substring(0, trimmed.IndexOf('}') + 1);

return end;

}

private static string GetValue(string value, string path2)

{

var path = string.Copy(path2);

var search = value;

var index = path.IndexOf(search, StringComparison.Ordinal);

var index2 = path.IndexOf(",", index, StringComparison.Ordinal);

if (index2 == -1)

index2 = path.IndexOf("}", index, StringComparison.Ordinal);

//Log.Fatal($"[{index}] [{search.Length}] [{index2}]");

var kek = path.Substring(index + search.Length, index2 - index - search.Length).Trim('\"');

if (kek == "null")

return 0.ToString();

return kek;

}

#endregion

}

}

//Log.Error("5");

}

try

{

possibleMmr = $"{estimate - stdDev}-{estimate + stdDev}";

}

catch (Exception)

{

//Log.Error("6");

}

try

{

matches = await FindMatches(steamId);

}

catch (Exception)

{

//Log.Error("7");

}

try

{

infoAboutHero = await FindInfoAboutHero(steamId, (uint)player.Hero.HeroId);

}

catch (Exception)

{

//Log.Error("8");

}

//Log.Debug("test: "+ matches);

Log.Debug(

$"[WinRate: {wr}] [solo: {solo}] [party {party}] [estimate mmr: {possibleMmr}] [{country}] history: {matches}");

string totalGames = "";

string wins = "";

try

{

totalGames = GetValue("games", infoAboutHero).TrimStart(':');

}

catch (Exception)

{

}

try

{

wins = GetValue("win", infoAboutHero).TrimStart(':');

}

catch (Exception)

{

}

var wrOnCurrentHero = 0;

try

{

wrOnCurrentHero = (int)((float)Convert.ToInt32(wins) / Convert.ToInt32(totalGames) \* 100.0f);

}

catch (Exception)

{

}

try

{

Log.Debug(

$"[Hero: {player?.Hero?.GetRealName()} -> [Games {totalGames}] [Wins {wins}] [WR {wrOnCurrentHero}%]");

}

catch (Exception)

{

}

PlayerInfoList.Add(new PlayerInfo((int)i, solo, party, country, possibleMmr, wr, matches, player?.Name, player?.Hero, totalGames, wins, wrOnCurrentHero));

}

catch (Exception e)

{

Log.Debug($"error with player: {player.Name} ({i}) -> {e}");

}

}

}

private static void Drawing\_OnEndScene(EventArgs args)

{

try

{

if (!Checker.IsActive()) return;

if (!Members.Menu.Item("OpenDota.Enable").GetValue<bool>()) return;

}

catch (Exception)

{

// ignored

}

if (Drawing.Direct3DDevice9 == null || Drawing.Direct3DDevice9.IsDisposed)

{

return;

}

if (Check)

{

var newlist = PlayerInfoList.ToList();

foreach (var playerInfo in newlist)

{

var id = playerInfo.Id;

var startXPosition = HeroPickStageScreenHelper.GetPlayerPosition(id);

var position = new Vector2(startXPosition, 35);

var size = HudInfoNew.GetTopPanelSizeY();

position += new Vector2(0, (float)size \* 1.8f);

var defClr = Color.White;

DrawShadowText(playerInfo.Name, (int)position.X, (int)position.Y, defClr);

position.Y += 15;

DrawShadowText(playerInfo.Wr, (int)position.X, (int)position.Y, defClr);

position.Y += 15;

DrawShadowText(

playerInfo.Solo == 0 ? $"Estimated: {playerInfo.PossibleMmr}" : $"Solo: {playerInfo.Solo}",

(int)position.X, (int)position.Y, defClr);

if (playerInfo.Party > 0)

{

position.Y += 15;

DrawShadowText($"Party: {playerInfo.Party}", (int)position.X, (int)position.Y, defClr);

}

var gameHistorySize = playerInfo.Matches.Length - 2;

if (gameHistorySize >= 1)

{

position.Y += 15;

for (var i = 0; i < gameHistorySize; i++)

{

var isTrue = playerInfo.Matches[i + 1] == '+';

var clr = isTrue ? Color.Green : Color.Red;

position.X += 10;

var text = '⬤';//●

DrawShadowText($"{text}", (int)position.X, (int)position.Y, clr);

}

}

if (playerInfo.Country.Length > 0)

{

try

{

var n = Convert.ToInt32(playerInfo.Country);

if (n > 0)

{

position.Y += 15;

DrawShadowText($"[{playerInfo.Country}]", (int) position.X, (int) position.Y, defClr);

}

}

catch (Exception)

{

}

}

if (playerInfo.TotalGames.Length > 0)

{

try

{

var n = Convert.ToInt32(playerInfo.TotalGames);

if (n == 0)

continue;

}

catch (Exception)

{

continue;

}

var totalGames = Convert.ToInt32(playerInfo.TotalGames);

var wins = Convert.ToInt32(playerInfo.Wins);

var loses = totalGames-wins;

var wr = playerInfo.WrOnCurrentHero;

position.Y += 15;

position.X = startXPosition;

DrawShadowText($"[{playerInfo.Hero?.GetRealName()}: {wins}/{loses} ({wr}%)]", (int)position.X, (int)position.Y, defClr);

}

}

}

}

#region Fakes

private static async void PartyFake()

{

Log.Debug("starting fake");

if (true)

{

\_loaded = true;

for (uint i = 0; i < 10; i++)

{

try

{

Printer.PrintSuccess(new string('-', Console.BufferWidth));

uint steamId = 105248644;

Log.Debug($"Player({i}): {"FAKE"} => id: {steamId}");

if (steamId <= 10)

{

Log.Error("Wrong steam id!");

continue;

}

var test = await FindWinRateAsync(steamId);

if (test < 0 || test > 100)

{

Log.Error("Cant load this player!");

continue;

}

var playerReq = await GetPlayerAsync(steamId);

var wr = await FindFullWinRateAsync(steamId);

//var accName = GetValue("personaname\":", playerReq);

int estimate = 0;

int stdDev = 0;

int solo = 0;

int party = 0;

string country = "";

string possibleMmr = "";

string matches = "";

/\*int estimate = Convert.ToInt32(GetValue("estimate\":", playerReq));

int stdDev = Convert.ToInt32(GetValue("stdDev\":", playerReq));

int solo = Convert.ToInt32(GetValue("solo\_competitive\_rank\":", playerReq));

int party = Convert.ToInt32(GetValue("competitive\_rank\":", playerReq));

var country = GetValue("loccountrycode\":", playerReq);

var possibleMmr = $"{estimate - stdDev}-{estimate + stdDev}";\*/

try

{

//Console.WriteLine("estimate: "+ GetValue("estimate\":", playerReq));

estimate = Convert.ToInt32(GetValue("{\"estimate\":", playerReq));

}

catch (Exception)

{

//Log.Error("1");

}

try

{

var item = GetValue("stdDev\":", playerReq);

item = item.Substring(0, item.IndexOf(".", StringComparison.Ordinal));

stdDev = Convert.ToInt32(item);

}

catch (Exception)

{

//Log.Error("2");

}

try

{

var item = GetValue("solo\_competitive\_rank\":", playerReq);

solo = Convert.ToInt32(item /\*.Substring(1, item.Length - 2)\*/);

}

catch (Exception)

{

//Log.Error("3");

}

try

{

var item = GetValue("\"competitive\_rank\":", playerReq);

party = Convert.ToInt32(item /\*.Substring(1, item.Length - 2)\*/);

}

catch (Exception)

{

//Log.Error("4");

}

try

{

country = GetValue("loccountrycode\":", playerReq);

}

catch (Exception)

{

//Log.Error("5");

}

try

{

possibleMmr = $"{estimate - stdDev}-{estimate + stdDev}";

}

catch (Exception)

{

//Log.Error("6");

}

try

{

matches = await FindMatches(steamId);

}

catch (Exception)

{

//Log.Error("7");

}

//Log.Debug("test: "+ matches);

PlayerInfoList.Add(new PlayerInfo((int)i, solo, party, country, possibleMmr, wr, matches,

"FAKE "+i));

Log.Debug(

$"[WinRate: {wr}] [solo: {solo}] [party {party}] [estimate mmr: {possibleMmr}] [{country}] history: {matches}");

//var success = await TryToFindPlayerAsync(player.Name);

//var success = TryToFindPlayer(player.Name);

/\*Log.Debug($"Try To Find player with search API -> player({i}): " + player.Name + " -> (steamId)" +

player.PlayerSteamID +

$" -> success: {success.Length != 2} bufferSize [{success.Length}]");

if (success.Length != 2)

{

var accId = GetValue("account\_id\":", success);

var accName = GetValue("personaname\":", success);

Log.Debug("id: " + accId);

Log.Debug("personaname: " + accName);

Log.Debug("wr: " + FindWinRate(Convert.ToUInt32(accId)) + "%");

//Log.Debug("GetPlayer: " + GetPlayer(accId));

}

else

{

Log.Debug($"cant find {player.Name}!");

}\*/

}

catch (Exception e)

{

Log.Debug($"error with player: ({i}) -> {e}");

}

}

}

Log.Debug("ending fake");

}

private static async void SingleFake()

{

Log.Debug("loading!");

var s = await GetPlayerAsync(1);

Log.Debug(s);

s = await FindInfoAboutHero(1, (uint) ObjectManager.LocalHero.HeroId);

Log.Debug(s);

var totalGames = GetValue("games", s).TrimStart(':');

var wins = GetValue("win", s).TrimStart(':');

var wrOnCurrentHero = (float)Convert.ToInt32(wins) / Convert.ToInt32(totalGames) \* 100.0f;

Log.Debug(

$"[Hero: {ObjectManager.LocalHero.GetRealName()} -> [Games {totalGames}] [Wins {wins}] [WR {wrOnCurrentHero}%]");

}

#endregion

#region render stuff

private static Vector2 DrawText(string text, Vector2 tSize, Vector2 startPos)

{

var textSize = Drawing.MeasureText(text, "Arial",

tSize, FontFlags.None);

Drawing.DrawRect(startPos, textSize, new Color(0, 0, 0, 155));

var textPos = startPos;

Drawing.DrawText(

text,

textPos, tSize,

Color.White,

FontFlags.AntiAlias | FontFlags.StrikeOut);

return textSize;

}

private static Vector2 DrawHeroIcon(Hero target, Vector2 size, Vector2 startPos)

{

var extra = new Vector2(size.X / 3, 0);

var finalSize = size + extra;

Drawing.DrawRect(startPos, finalSize, Textures.GetHeroTexture(target.StoredName()));

Drawing.DrawRect(startPos, finalSize, new Color(0, 0, 0, 255), true);

return finalSize;

}

private static void DrawShadowText(string stext, int x, int y, Color clr)

{

Text.DrawText(null, stext, x + 1, y + 1, Color.Black);

Text.DrawText(null, stext, x, y, clr);

}

private static void CurrentDomainDomainUnload(object sender, EventArgs e)

{

try { if (!Checker.IsActive()) return; } catch { }

Text?.Dispose();

}

private static void Drawing\_OnPostReset(EventArgs args)

{

try { if (!Checker.IsActive()) return; } catch { }

Text?.OnLostDevice();

}

private static void Drawing\_OnPreReset(EventArgs args)

{

try { if (!Checker.IsActive()) return; } catch { }

Text?.OnLostDevice();

}

#endregion

#region Helpers

private static string TryToFindPlayer(string name, bool print = false)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/search?q={name}&similarity=1");

string strContent;

using (var response = webRequest.GetResponse())

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

if (print)

Console.WriteLine(strContent);

}

return strContent;

}

private static async Task<string> TryToFindPlayerAsync(string name, bool print = false)

{

var request = WebRequest.Create($"https://api.opendota.com/api/search?q={name}&similarity=1");

string strContent;

using (var response = (HttpWebResponse)await Task.Factory

.FromAsync(request.BeginGetResponse,

request.EndGetResponse,

null))

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

if (print)

Console.WriteLine(strContent);

}

return strContent;

}

private static string GetPlayer(string id)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{id}");

string strContent;

using (var response = webRequest.GetResponse())

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

//Console.WriteLine(strContent);

}

return strContent;

}

private static async Task<string> GetPlayerAsync(uint id)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{id}");

string strContent;

var response = (HttpWebResponse)await Task.Factory

.FromAsync(webRequest.BeginGetResponse,

webRequest.EndGetResponse,

null);

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

//Console.WriteLine(strContent);

}

return strContent;

}

private static int FindWinRate(uint id)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{id}/wl");

//((HttpWebRequest)webRequest).UserAgent = ".NET Framework";

string strContent;

using (var response = webRequest.GetResponse())

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

//Console.WriteLine(strContent);

}

int win = Convert.ToInt32(GetValue("win\":", strContent));

int lose = Convert.ToInt32(GetValue("lose\":", strContent));

return (int)(win / (win + (double)lose) \* 100f);

}

private static async Task<int> FindWinRateAsync(uint id)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{id}/wl");

//((HttpWebRequest)webRequest).UserAgent = ".NET Framework";

string strContent;

var response = (HttpWebResponse)await Task.Factory

.FromAsync(webRequest.BeginGetResponse,

webRequest.EndGetResponse,

null);

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

//Console.WriteLine(strContent);

}

int win = Convert.ToInt32(GetValue("win\":", strContent));

int lose = Convert.ToInt32(GetValue("lose\":", strContent));

return (int)(win / (win + (double)lose) \* 100f);

}

private static async Task<string> FindFullWinRateAsync(uint id)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{id}/wl");

//((HttpWebRequest)webRequest).UserAgent = ".NET Framework";

string strContent;

var response = (HttpWebResponse)await Task.Factory

.FromAsync(webRequest.BeginGetResponse,

webRequest.EndGetResponse,

null);

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

}

int win = Convert.ToInt32(GetValue("win\":", strContent));

int lose = Convert.ToInt32(GetValue("lose\":", strContent));

var wr = (int)(win / (win + (double)lose) \* 100f);

return $"({win}/{lose}) {wr}%";

}

private static async Task<string> FindMatches(uint id, int gameLimit = 5, int gameMode = 22)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{id}/matches?limit={gameLimit}&game\_mode={gameMode}");

//((HttpWebRequest)webRequest).UserAgent = ".NET Framework";

var response = (HttpWebResponse)await Task.Factory

.FromAsync(webRequest.BeginGetResponse,

webRequest.EndGetResponse,

null);

var info = "[";

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

var strContent = reader.ReadToEnd();

for (var i = 0; i < gameLimit; i++)

{

var start = strContent.IndexOf("{", StringComparison.Ordinal);

var end = strContent.IndexOf("}", StringComparison.Ordinal);

if (start == -1 || end == -1)

continue;

var tempLine = strContent.Substring(start, end);

strContent = strContent.Remove(start, end);

//Console.WriteLine(new string('-', Console.BufferWidth));

//Console.WriteLine(tempLine);

var playerSlot = Convert.ToInt32(GetValue("player\_slot\":", tempLine));

var radiantWin = GetValue("radiant\_win\":", tempLine) == "true";

var win = (playerSlot <= 10 && radiantWin) || (playerSlot > 10 && !radiantWin);

info += win ? "+" : "-";

}

}

info += "]";

return info;

}

private static async Task<string> FindInfoAboutHero(uint playerid,uint heroId)

{

var webRequest = WebRequest.Create($"https://api.opendota.com/api/players/{playerid}/heroes?hero\_id={heroId}");

string strContent;

var response = (HttpWebResponse)await Task.Factory

.FromAsync(webRequest.BeginGetResponse,

webRequest.EndGetResponse,

null);

using (var content = response.GetResponseStream())

using (var reader = new StreamReader(content))

{

strContent = reader.ReadToEnd();

//Console.WriteLine(strContent);

}

string trimmed = strContent.Trim();

var end = trimmed.Substring(0, trimmed.IndexOf('}') + 1);

return end;

}

private static string GetValue(string value, string path2)

{

var path = string.Copy(path2);

var search = value;

var index = path.IndexOf(search, StringComparison.Ordinal);

var index2 = path.IndexOf(",", index, StringComparison.Ordinal);

if (index2 == -1)

index2 = path.IndexOf("}", index, StringComparison.Ordinal);

//Log.Fatal($"[{index}] [{search.Length}] [{index2}]");

var kek = path.Substring(index + search.Length, index2 - index - search.Length).Trim('\"');

if (kek == "null")

return 0.ToString();

return kek;

}

#endregion

}

}

namespace OverlayInformation

{

public class Holder : IDisposable

{

public List<AbilityHolder> Holders;

public Holder()

{

Holders = new List<AbilityHolder>();

UpdateManager.Subscribe(Flush, 5000);

}

private void Flush()

{

var temp = Holders.Where(x => x.IsValid);

Holders = temp.ToList();

}

public AbilityHolder GetOrCreate(Ability ability)

{

var find = Holders.Find(x => x.IsValid && x.Ability.Equals(ability));

if (find != null) return find;

find = new AbilityHolder(ability);

Holders.Add(find);

return find;

}

public void Dispose()

{

UpdateManager.Unsubscribe(Flush);

}

}

public class AbilityHolder

{

private static readonly ILog Log = AssemblyLogs.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

public Ability Ability;

public uint Handle;

public Hero Owner;

public AbilityId Id;

public AbilityState AbilityState;

public DotaTexture Texture;

public float Cooldown;

public bool IsUltimate;

public bool IsHidden;

public bool IsValid => Ability != null && Ability.IsValid;

public int MaximumLevel { get; set; }

public AbilitySlot AbilitySlot { get; set; }

public string Name { get; set; }

public Item Item;

public uint Cost;

public AbilityHolder(Ability ability)

{

Ability = ability;

Handle = ability.Handle;

Name = ability.Name;

MaximumLevel = Ability.MaximumLevel;

Owner = (Hero)ability.Owner;

Id = Ability.Id;

AbilityState = ability.AbilityState;

Texture = Ability is Item

? Textures.GetItemTexture(ability.Name)

: Textures.GetSpellTexture(ability.Name);

Cooldown = Ability.Cooldown;

IsUltimate = ability.AbilityType == AbilityType.Ultimate;

IsHidden = ability.IsHidden;

AbilitySlot = ability.AbilitySlot;

Item = ability as Item;

if (Item != null)

{

Cost = Item.Cost;

}

UpdateManager.BeginInvoke(async () =>

{

while (ability.IsValid)

{

AbilityState = ability.AbilityState;

Cooldown = Ability.Cooldown;

IsHidden = ability.IsHidden;

AbilitySlot = ability.AbilitySlot;

await Task.Delay(300);

}

//Log.Debug($"[{Owner.Name}] end for -> {Id}");

});

}

}

public class CourContainer : IDisposable

{

public Courier Cour { get; }

public bool IsAlly { get; }

public OverlayInformation Main { get; }

public CourContainer(Courier cour, bool isAlly, OverlayInformation main)

{

Cour = cour;

IsAlly = isAlly;

Main = main;

Items = new List<Item>();

UpdateManager.Subscribe(UpdateItems, 500);

UpdateManager.Subscribe(FlushChecker, 1000);

}

private void FlushChecker()

{

if (Cour == null || !Cour.IsValid)

{

Dispose();

}

}

private void UpdateItems()

{

Items = Cour.Inventory.Items.ToList();

}

public List<Item> Items { get; set; }

public void Dispose()

{

UpdateManager.Unsubscribe(UpdateItems);

UpdateManager.Unsubscribe(FlushChecker);

}

}

public class HeroContainer

{

public static string GamePath = Game.GamePath;

private static readonly ILog Log = AssemblyLogs.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

public bool IsAlly { get; }

public OverlayInformation Main { get; }

public bool IsOwner { get; }

public Hero Hero { get; }

//public Ability Ultimate;

public AbilityHolder Ultimate;

public List<AbilityHolder> Abilities2;

public List<AbilityHolder> Items;

public List<AbilityHolder> DangItems;

public List<AbilityHolder> InvisBreakerItems;

public float LastTimeUnderVision;

public float Health;

public float MaxHealth;

public float Mana;

public float MaxMana;

public bool IsVisible;

public uint Networth;

public bool DontDraw;

public bool AghanimState;

public Inventory HeroInventory { get; set; }

private static readonly List<AbilityId> DangeItemList = new List<AbilityId>

{

AbilityId.item\_blink,

AbilityId.item\_gem,

AbilityId.item\_silver\_edge,

AbilityId.item\_sheepstick,

AbilityId.item\_orchid,

AbilityId.item\_bloodthorn,

AbilityId.item\_black\_king\_bar,

AbilityId.item\_glimmer\_cape,

AbilityId.item\_invis\_sword

};

private static readonly List<AbilityId> InvisBreakerList = new List<AbilityId>

{

AbilityId.item\_gem,

AbilityId.item\_dust,

AbilityId.item\_ward\_sentry,

AbilityId.item\_ward\_dispenser,

};

//private readonly InventoryManager \_manager;

public float TimeInFog;

public AbilityState AbilityState { get; set; }

public Holder HolderHelper;

public string Name { get; set; }

public List<Ability> GetAllAbilities => Hero.Spellbook.Spells.Where(

x => x.Name != "generic\_hidden" && (x.AbilityType == AbilityType.Basic || x.AbilityType == AbilityType.Ultimate))

.ToList();

public HeroContainer(Hero hero, bool isAlly, OverlayInformation main)

{

var itemString = hero.HeroId.ToString().Remove(0, 14);

main.Context.Value.TextureManager.LoadFromFile(hero.HeroId.ToString(),

$@"{GamePath}\game\dota\materials\ensage\_ui\miniheroes\png\{itemString}.png");

/\*Log.Warn($"Texture Name: {itemString}");

Log.Warn($"GamePath: {$@"{GamePath}\game\dota\materials\ensage\_ui\miniheroes\png\{itemString}.png"}");\*/

//$@"resource\flash3\images\heroes\miniheroes\{hero.HeroId}.png");

Name = hero.Name;

Id = hero.Player == null ? 0 : hero.Player.Id;

HolderHelper = new Holder();

IsAlly = isAlly;

Main = main;

IsOwner = hero.Equals(ObjectManager.LocalHero);

Hero = hero;

//Ultimate = hero.Spellbook.Spells.First(x => x.AbilityType == AbilityType.Ultimate);

LastTimeUnderVision = Game.RawGameTime;

Items = new List<AbilityHolder>();

DangItems = new List<AbilityHolder>();

InvisBreakerItems = new List<AbilityHolder>();

Abilities2 = new List<AbilityHolder>();

foreach (var ability in GetAllAbilities)

{

var holder = HolderHelper.GetOrCreate(ability);//new AbilityHolder(ability);

Abilities2.Add(holder);

if (holder.IsUltimate)

Ultimate = holder;

Log.Info($"{ability.Name} -> {(ability.AbilityType == AbilityType.Basic ? "basic" : "ultimate")}");

}

HeroInventory = Hero.Inventory;

/\*\_manager = new InventoryManager(new EnsageServiceContext(hero));

//manager.CollectionChanged += ManagerOnCollectionChanged;

\_manager.CollectionChanged += (sender, args) =>

{

//Items.Clear();

DangItems.Clear();

Items = \_manager.Inventory.Items.ToList();

Networth = 0;

var tmpAgh = hero.HasAghanimsScepter();

if (!AghanimState && tmpAgh || AghanimState && !tmpAgh)

{

RefreshAbilities();

}

AghanimState = tmpAgh;

foreach (var item in Items)

{

Networth += item.Cost;

try

{

if (DangeItemList.Contains(item.Id))

DangItems.Add(item);

}

catch (Exception e)

{

Console.WriteLine("GEGE -> "+e);

}

}

};\*/

UpdateInfo();

UpdateItems();

UpdateManager.Subscribe(UpdateItems, 500);

UpdateManager.Subscribe(UpdateInfo, 250);

UpdateManager.Subscribe(FlushChecker,1000);

var dividedWeStand = hero.Spellbook.SpellR as DividedWeStand;

if (dividedWeStand != null && hero.HeroId == HeroId.npc\_dota\_hero\_meepo && dividedWeStand.UnitIndex > 0)

{

DontDraw = true;

}

HeroId = hero.HeroId;

if (HeroId == HeroId.npc\_dota\_hero\_rubick || HeroId == HeroId.npc\_dota\_hero\_doom\_bringer/\* ||

classId == ClassId.CDOTA\_Unit\_Hero\_Invoker\*/ || HeroId == HeroId.npc\_dota\_hero\_morphling)

{

UpdateManager.Subscribe(AbilityUpdater, 750);

}

/\*Main.Context.Value.AbilityDetector.AbilityCasted += (sender, args) =>

{

Game.PrintMessage(args.Ability.Ability.Name);

};

Main.Context.Value.AbilityDetector.AbilityCastStarted += (sender, args) =>

{

Game.PrintMessage(args.Ability.Ability.Name);

};\*/

}

public HeroId HeroId { get; set; }

private void FlushChecker()

{

if (Hero == null || !Hero.IsValid)

{

if (Game.GameState == GameState.GameInProgress)

{

var player = ObjectManager.GetPlayerById((uint) Id);

Log.Error(

$"CUSTOM FLUSH FOR {Name} id -> [{Id}] -> player -> [{(player != null ? player.Name : "null")}]");

}

Flush();

}

else if (Hero.IsIllusion && !Hero.HasModifier("modifier\_morphling\_replicate") && Hero.IsAlive && Hero.IsVisible)

{

Log.Error(

$"Flush cuz illusion {Name} id -> [{Id}]");

Flush();

}

}

private void UpdateInfo()

{

if (Hero == null || !Hero.IsValid)

return;

IsVisible = Hero.IsVisible;

if (IsVisible)

LastTimeUnderVision = Game.RawGameTime;

else

{

TimeInFog = Game.RawGameTime - LastTimeUnderVision;

return;

}

Health = Hero.Health;

Mana = Hero.Mana;

MaxHealth = Hero.MaximumHealth;

MaxMana = Hero.MaximumMana;

if (Ultimate != null && Ultimate.IsValid)

AbilityState = Ultimate.AbilityState;

}

private void UpdateItems()

{

if (Hero == null || !Hero.IsValid)

return;

if (!Hero.IsAlive || !Hero.IsVisible)

return;

DangItems.Clear();

InvisBreakerItems.Clear();

Items = new List<AbilityHolder>();

Networth = 0;

foreach (var item in HeroInventory.Items)

{

var localHolder = HolderHelper.GetOrCreate(item);

Items.Add(localHolder);

Networth += item.Cost;

if (Main.Config.HeroOverlay.ItemDangItems)

{

if (DangeItemList.Contains(item.Id))

DangItems.Add(localHolder);

}

else if (Main.Config.HeroOverlay.ItemInvisBreakItems)

{

if (InvisBreakerList.Contains(item.Id))

InvisBreakerItems.Add(localHolder);

}

}

var tmpAgh = Hero.HasAghanimsScepter();

if (!AghanimState && tmpAgh || AghanimState && !tmpAgh)

{

RefreshAbilities2();

}

AghanimState = tmpAgh;

}

public int Id { get; set; }

private void AbilityUpdater()

{

//var needToRefresh = Abilities.Any(x => x == null || !x.IsValid || x.IsHidden);

var needToRefresh = Abilities2.Any(x => x == null || !x.IsValid || x.IsHidden);

if (needToRefresh)

{

//Game.PrintMessage($"need to rrefresh for {this.HeroId}");

RefreshAbilities2();

}

}

/\*private void ManagerOnCollectionChanged(object sender, NotifyCollectionChangedEventArgs args)

{

if (args.Action == NotifyCollectionChangedAction.Add)

{

foreach (InventoryItem iitem in args.NewItems)

{

Networth += iitem.Item.Cost;

if (DangeItemList.Contains(iitem.Id))

DangItems.Add(iitem.Item);

Items.Add(iitem.Item);

}

}

else if (args.Action == NotifyCollectionChangedAction.Remove)

{

foreach (InventoryItem iitem in args.OldItems)

{

Networth -= iitem.Item.Cost;

if (DangeItemList.Contains(iitem.Id))

DangItems.Remove(iitem.Item);

Items.Remove(iitem.Item);

}

}

}\*/

public void RefreshAbilities2()

{

var abilities =

Hero.Spellbook.Spells.Where(

x =>

(x.AbilityType == AbilityType.Basic || x.AbilityType == AbilityType.Ultimate) &&

Abilities2.Find(y => y.Handle == x.Handle) == null && !x.IsHidden);

foreach (var ability in abilities)

{

Abilities2.Add(HolderHelper.GetOrCreate(ability));

//Abilities2.Add(new AbilityHolder(ability));

Log.Info($"added new ability -> {ability.Name} ({ability.Owner.Name})");

//Game.PrintMessage($"added new ability -> {ability.Name} ({ability.Owner.Name})");

}

Abilities2.RemoveAll(x => !x.IsValid/\* || x.IsHidden\*/);

}

public void Flush()

{

//\_manager.Deactivate();

if (HeroId == HeroId.npc\_dota\_hero\_rubick || HeroId == HeroId.npc\_dota\_hero\_doom\_bringer /\* ||

classId == ClassId.CDOTA\_Unit\_Hero\_Invoker\*/ || HeroId == HeroId.npc\_dota\_hero\_morphling)

{

UpdateManager.Unsubscribe(AbilityUpdater);

}

UpdateManager.Unsubscribe(UpdateItems);

UpdateManager.Unsubscribe(UpdateInfo);

UpdateManager.Unsubscribe(FlushChecker);

HolderHelper.Dispose();

}

}

}namespace OverlayInformation

{

public static class HudInfo

{

#region Static Fields

/// <summary>

/// The dire compare.

/// </summary>

public static double DireCompare;

/// <summary>

/// The health bar height.

/// </summary>

private static readonly double HpBarHeight;

/// <summary>

/// The health bar width.

/// </summary>

private static readonly double HpBarWidth;

/// <summary>

/// The health bar x.

/// </summary>

private static readonly double HpBarX;

/// <summary>

/// The health bar y.

/// </summary>

private static readonly float HpBarY;

/// <summary>

/// The monitor.

/// </summary>

internal static readonly float Monitor;

/// <summary>

/// The player id dictionary.

/// </summary>

private static readonly Dictionary<float, int> PlayerIdDictionary = new Dictionary<float, int>();

/// <summary>

/// The radiant compare.

/// </summary>

public static double RadiantCompare;

/// <summary>

/// The rate.

/// </summary>

private static readonly float Rate;

/// <summary>

/// The screen size.

/// </summary>

private static readonly Vector2 ScreenSize;

/// <summary>

/// The x.

/// </summary>

internal static double X;

/// <summary>

/// The y.

/// </summary>

private static double y;

#endregion

#region Constructors and Destructors

/// <summary>

/// Initializes static members of the <see cref="HUDInfo" /> class.

/// </summary>

static HudInfo()

{

double tinfoHeroDown;

double panelHeroSizeX;

float compareWidth;

ScreenSize = new Vector2(Drawing.Width, Drawing.Height);

if (ScreenSize.X == 0)

{

Console.WriteLine("Ensage couldnt determine your resolution, try to launch in window mode");

return;

}

var ratio = Math.Floor((decimal)(ScreenSize.X / ScreenSize.Y \* 100));

if (ratio == 213)

{

compareWidth = 1600;

panelHeroSizeX = 45.28;

tinfoHeroDown = 25.714;

DireCompare = 2.402;

RadiantCompare = 3.08;

HpBarHeight = 7;

HpBarWidth = 69;

HpBarX = 36;

HpBarY = 23;

}

else if (ratio == 177)

{

compareWidth = 1600;

panelHeroSizeX = 52.8900000000004;

tinfoHeroDown = 25.714;

DireCompare = 2.5001;

RadiantCompare = 3.409;

HpBarHeight = 10;

HpBarWidth = 84;

HpBarX = 43;

HpBarY = 27;

}

else if (ratio == 166)

{

compareWidth = 1280;

panelHeroSizeX = 47.19;

tinfoHeroDown = 25.714;

DireCompare = 2.59;

RadiantCompare = 3.64;

HpBarHeight = 7.4;

HpBarWidth = 71;

HpBarX = 37;

HpBarY = 22;

}

else if (ratio == 160)

{

compareWidth = 1280;

panelHeroSizeX = 48.95;

tinfoHeroDown = 25.714;

DireCompare = 2.609;

RadiantCompare = 3.78;

HpBarHeight = 9;

HpBarWidth = 75;

HpBarX = 38.3;

HpBarY = 25;

}

else if (ratio == 150)

{

compareWidth = 1280;

panelHeroSizeX = 51.39;

tinfoHeroDown = 25.714;

DireCompare = 2.64;

RadiantCompare = 4.02;

HpBarHeight = 8;

HpBarWidth = 79.2;

HpBarX = 40.2;

HpBarY = 24;

}

else if (ratio == 133)

{

compareWidth = 1024;

panelHeroSizeX = 47.21;

tinfoHeroDown = 25.714;

DireCompare = 2.775;

RadiantCompare = 4.57;

HpBarHeight = 8;

HpBarWidth = 71;

HpBarX = 36.6;

HpBarY = 23;

}

else if (ratio == 125)

{

compareWidth = 1280;

panelHeroSizeX = 58.3;

tinfoHeroDown = 25.714;

DireCompare = 2.78;

RadiantCompare = 4.65;

HpBarHeight = 11;

HpBarWidth = 96.5;

HpBarX = 49;

HpBarY = 32;

}

else

{

Console.WriteLine(

@"Your screen resolution is not supported and drawings might have wrong size/position, (" + ratio

+ ")");

compareWidth = 1600;

panelHeroSizeX = 65;

tinfoHeroDown = 25.714;

DireCompare = 2.655;

RadiantCompare = 5.985;

HpBarHeight = 10;

HpBarWidth = 83.5;

HpBarX = 43;

HpBarY = 28;

}

Monitor = ScreenSize.X / compareWidth;

Rate = Math.Max(Monitor, 1);

X = panelHeroSizeX \* Monitor;

y = ScreenSize.Y / tinfoHeroDown;

//Drawing.OnDraw += Drawing\_OnDraw;

//var mipos = new Vector3(MapLeft, MapTop, 0).WorldToMinimap();

//var minimap = new Render.Rectangle(

// mipos.X,

// mipos.Y,

// currentMinimap.Size.X,

// currentMinimap.Size.Y,

// new ColorBGRA(100, 100, 100, 50));

//minimap.Add();

/\*\_line = new Line(Drawing.Direct3DDevice9);

Drawing.OnEndScene += args =>

{

if (Drawing.Direct3DDevice9 == null)

{

return;

}

var pos = currentMinimap.Position+new Vector2(0,800);

var size = currentMinimap.Size;

DrawLine(

pos.X,

pos.Y,

pos.X + size.X,

pos.Y + size.Y,

2, Color.YellowGreen);

};

Drawing.OnPostReset += args =>

{

\_line.OnResetDevice();

};

Drawing.OnPreReset += args =>

{

\_line.OnLostDevice();

};\*/

}

/\*public static void DrawLine(float x1, float y1, float x2, float y2, float w, Color color)

{

var vLine = new[] { new Vector2(x1, y1), new Vector2(x2, y2) };

\_line.GLLines = true;

\_line.Antialias = false;

\_line.Width = w;

\_line.Begin();

\_line.Draw(vLine, color);

\_line.End();

}\*/

//private static Line \_line;

#endregion

#region Public Methods and Operators

/// <summary>

/// Returns HealthBar position for given unit

/// </summary>

/// <param name="unit">

/// The unit.

/// </param>

/// <returns>

/// The <see cref="Vector2" />.

/// </returns>

public static Vector2 GetHPbarPosition(Unit unit)

{

var pos = unit.Position + new Vector3(0, 0, unit.HealthBarOffset);

Vector2 screenPos;

if (!Drawing.WorldToScreen(pos, out screenPos))

{

return Vector2.Zero;

}

var localHero = ObjectManager.LocalHero;

if (localHero != null && Equals(unit, localHero))

{

if (unit.ClassId == ClassId.CDOTA\_Unit\_Hero\_Meepo)

{

return screenPos + new Vector2((float)(-HpBarX \* 1.05 \* Monitor), (float)(-HpBarY \* 1.3 \* Monitor));

}

return screenPos + new Vector2((float)(-HpBarX \* 1.05 \* Monitor), (float)(-HpBarY \* 1.15 \* Monitor));

}

return screenPos + new Vector2((float)(-HpBarX \* Monitor), -HpBarY \* Monitor);

}

/// <summary>

/// Returns HealthBar X position for given unit

/// </summary>

/// <param name="unit">

/// The unit.

/// </param>

/// <returns>

/// The <see cref="float" />.

/// </returns>

public static float GetHPBarSizeX(Unit unit = null)

{

var hero = ObjectManager.LocalHero;

if (unit != null && hero != null && Equals(unit, hero))

{

return (float)((float)HpBarWidth \* Monitor \* 1.1);

}

return (float)HpBarWidth \* Monitor;

}

/// <summary>

/// Returns HealthBar Y position for given unit

/// </summary>

/// <param name="unit">

/// The unit.

/// </param>

/// <returns>

/// The <see cref="float" />.

/// </returns>

public static float GetHpBarSizeY(Unit unit = null)

{

var hero = ObjectManager.LocalHero;

if (unit != null && hero != null && Equals(unit, hero))

{

return (float)(HpBarHeight \* Monitor \* 1.05);

}

return (float)(HpBarHeight \* Monitor);

}

/// <summary>

/// Returns top panel position for given hero

/// </summary>

/// <param name="hero">

/// The hero.

/// </param>

/// <returns>

/// The <see cref="Vector2" />.

/// </returns>

public static Vector2 GetTopPanelPosition(Hero hero)

{

int id;

if (hero.Player == null)

{

if (PlayerIdDictionary.ContainsKey(hero.Handle))

{

id = PlayerIdDictionary[hero.Handle];

}

else

{

return Vector2.Zero;

}

}

else

{

id = hero.Player.Id;

}

if (!PlayerIdDictionary.ContainsKey(hero.Handle))

{

PlayerIdDictionary.Add(hero.Handle, id);

}

else

{

PlayerIdDictionary[hero.Handle] = id;

}

return new Vector2((float)(GetXX(hero) - 20 \* Monitor + X \* id), 0);

}

private class FakeClass

{

public readonly int Id;

public readonly Team Team;

public FakeClass(int id, Team team)

{

Id = id;

Team = team;

}

}

private static readonly Dictionary<FakeClass, Vector2> FakeDict = new Dictionary<FakeClass, Vector2>();

public static Vector2 GetFakeTopPanelPosition(int id, Team team)

{

var fake = FakeDict.Find(x => x.Key.Id == id && x.Key.Team == team);

if (!fake.Value.IsZero)

{

return fake.Value;

}

else

{

var pos = new Vector2((float) (GetFakeXX(team) - 20 \* Monitor + X \* id), 0);

FakeDict.Add(new FakeClass(id,team), pos);

return pos;

}

return new Vector2((float)(GetFakeXX(team) - 20 \* Monitor + X \* id), 0);

}

/// <summary>

/// Returns top panel size

/// </summary>

/// <param name="hero">

/// The hero.

/// </param>

/// <returns>

/// The <see cref="double[]" />.

/// </returns>

public static Vector2 GetTopPanelSize(Hero hero = null)

{

var size = new Vector2((float)GetTopPanelSizeX(hero), (float)GetTopPanelSizeY(hero));

return size;

}

/// <summary>

/// Returns top panel hero icon width

/// </summary>

/// <param name="hero">

/// The hero.

/// </param>

/// <returns>

/// The <see cref="double" />.

/// </returns>

public static double GetTopPanelSizeX(Hero hero = null)

{

return X;

}

/// <summary>

/// Returns top panel hero icon height

/// </summary>

/// <param name="hero">

/// The hero.

/// </param>

/// <returns>

/// The <see cref="double" />.

/// </returns>

public static double GetTopPanelSizeY(Hero hero = null)

{

return 35 \* Rate;

}

/// <summary>

/// The ratio percentage.

/// </summary>

/// <returns>

/// The <see cref="float" />.

/// </returns>

public static float RatioPercentage()

{

return Monitor;

}

/// <summary>

/// Returns screen width

/// </summary>

/// <returns>

/// The <see cref="float" />.

/// </returns>

public static float ScreenSizeX()

{

return ScreenSize.X;

}

/// <summary>

/// Returns screen height

/// </summary>

/// <returns>

/// The <see cref="float" />.

/// </returns>

public static float ScreenSizeY()

{

return ScreenSize.Y;

}

#endregion

#region Methods

private static void Drawing\_OnDraw(EventArgs args)

{

Drawing.DrawRect(GetTopPanelPosition(ObjectManager.LocalHero), GetTopPanelSize(ObjectManager.LocalHero), Color.White);

var v = ObjectManager.LocalHero;

/\*for (int i = 0; i < 5; i++)

{

Drawing.DrawRect(GetFakeTopPanelPosition(i, Team.Radiant), GetTopPanelSize(v), Color.White);

}\*/

/\*for (int i = 5; i < 10; i++)

{

Drawing.DrawRect(GetFakeTopPanelPosition(i, Team.Dire), GetTopPanelSize(v), Color.White);

}\*/

Drawing.DrawRect(GetFakeTopPanelPosition(0, Team.Radiant), GetTopPanelSize(v), Color.White);

Drawing.DrawRect(GetFakeTopPanelPosition(9, Team.Dire), GetTopPanelSize(v), Color.White);

Drawing.DrawRect(GetFakeTopPanelPosition(7, Team.Dire), GetTopPanelSize(v), Color.White);

Drawing.DrawRect(GetFakeTopPanelPosition(5, Team.Dire), GetTopPanelSize(v), Color.White);

}

/// <summary>

/// The get xx.

/// </summary>

/// <param name="hero">

/// The hero.

/// </param>

/// <returns>

/// The <see cref="double" />.

/// </returns>

private static double GetXX(Entity hero)

{

var screenSize = new Vector2(Drawing.Width, Drawing.Height);

if (hero.Team == Team.Radiant)

{

return screenSize.X / RadiantCompare + 1;

}

return screenSize.X / DireCompare + 1;

}

private static double GetFakeXX(Team team)

{

var screenSize = new Vector2(Drawing.Width, Drawing.Height);

if (team == Team.Radiant)

{

return screenSize.X / RadiantCompare + 1;

}

return screenSize.X / DireCompare + 1;

}

#endregion

}

}