SprintThree

February 16, 2025

1 Semester 2, Sprint 3: FULL DUELS MODE, and Bug Fixes, Bug Fixes...

Owen Kroeger

- Video Link -

- Jira Link -

- GitHub Link - Full Duels mode capability was implemented: Bot now plays a full versus game all the way through and detects when the game ends. Fixed many bugs including coordinate calibration, game chat interfering with gameplay, transitions not being detected, end screen not being detected.

Key Objectives: - Fully Implement Duels mode - Finalize gameplay element so focus can permanently shift to solely improving model

1.1 COMPLETE Duels Mode Implementation

1.1.1 What is Duels Mode?

Duels Mode is a competitive **GeoGuessr** format where two players receive the same location and compete to guess closer to the actual spot. The player with the more accurate guess deals damage to their opponent, and the game continues until one player's health reaches zero.

1.1.2 Evolution of the Duels Bot

The first iteration of the **Duels Bot** played exactly **5 rounds**, mimicking a standard solo game format. However, this was not fully aligned with how **Duels Mode** actually works, since Duels continue dynamically until a player wins.

1.1.3 Smarter Game Detection

To improve the bot's performance, it now **dynamically detects when a game has ended**, rather than assuming a fixed number of rounds. This is achieved using **image comparison techniques**: 1. At the start of a game, the bot captures a **reference screenshot** of the **end screen**. 2. At the end of each round, it captures another **real-time screenshot** of the end screen. 3. The bot then compares the two images using **structural similarity metrics (SSIM)** instead of simple pixel-by-pixel differences. 4. If the similarity is high, it determines that the **game has ended**, allowing the bot to gracefully stop.

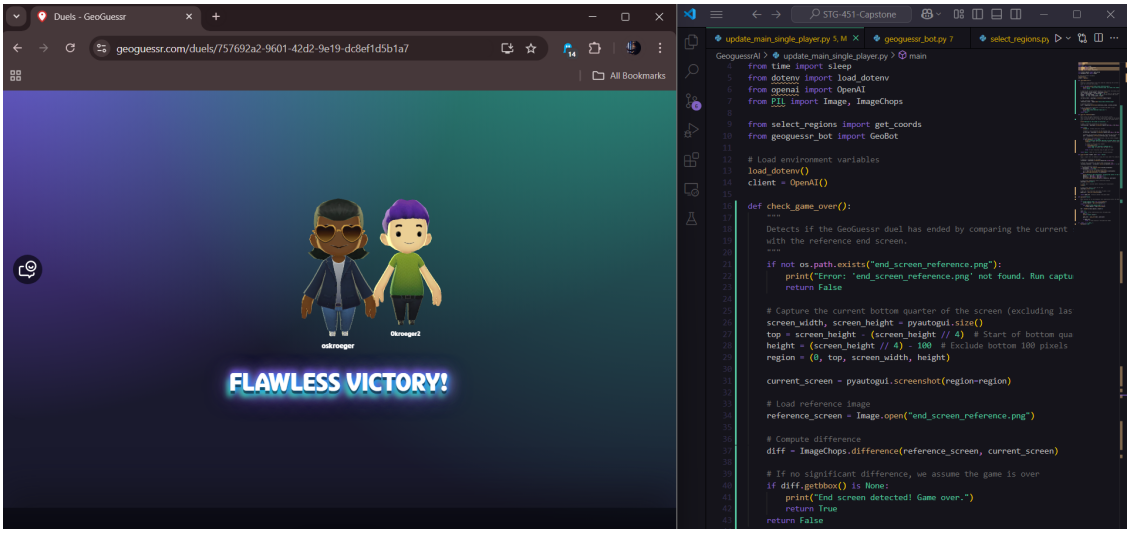
1.1.4 Why is this Important?

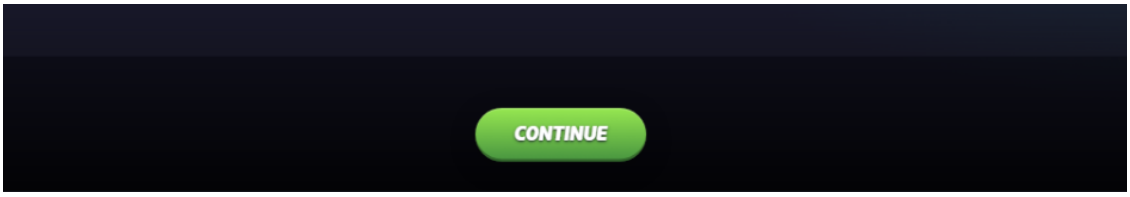
- **No More Fixed Rounds:** The bot no longer assumes a 5-round limit—it plays until the match **naturally concludes**.
- **More Accurate End Detection:** Instead of relying on abrupt screen changes, the bot detects the **actual transition** between gameplay and results.
- **Better Performance & Stability:** Using **SSIM-based image comparison** reduces false positives caused by UI changes or minor screen differences.

With this enhancement, the bot now behaves much closer to how a human would perceive the **end of a duel**, making it more reliable for **automated gameplay**.

[ ]:
```
def check_game_over():
    """
    Detects if the GeoGuessr duel has ended by comparing the current screen
    with the reference end screen.
    """
    if not os.path.exists("end_screen_reference.png"):
        print("Error: 'end_screen_reference.png' not found. Run capture script␣
↪first.")
        return False

    # Capture the current bottom quarter of the screen (excluding last 100␣
↪pixels)
    screen_width, screen_height = pyautogui.size()
    top = screen_height - (screen_height // 4)  # Start of bottom quarter
    height = (screen_height // 4) - 100  # Exclude bottom 100 pixels
    region = (0, top, screen_width, height)

    current_screen = pyautogui.screenshot(region=region)

    # Load reference image
    reference_screen = Image.open("end_screen_reference.png")

    # Compute difference
    diff = ImageChops.difference(reference_screen, current_screen)

    # If no significant difference, we assume the game is over
    if diff.getbbox() is None:
        print("End screen detected! Game over.")
        return True
    return False

def wait_for_transition(bot):
    """
    Waits until the game transitions to the results screen.
    This is determined by detecting a significant change in the minimap area.
    After waiting 15 seconds, it checks if the game has ended before continuing.
```

```
    """
    print("Waiting for the round to transition...")

    # Take a reference screenshot of the minimap
    reference_map = pyautogui.screenshot(region=(bot.map_x, bot.map_y, bot.
↪map_w, bot.map_h))

    while True:
        sleep(0.5)  # Check every 0.5 seconds

        # Capture a new screenshot of the minimap area
        current_map = pyautogui.screenshot(region=(bot.map_x, bot.map_y, bot.
↪map_w, bot.map_h))

        # Compute the difference between the reference and current image
        diff = ImageChops.difference(reference_map, current_map)

        # If there's a noticeable difference, the map likely changed␣
↪(transition detected)
        if diff.getbbox() is not None:
            print("Transition detected! Waiting 15 seconds before checking if␣
↪game ended...")
            sleep(15)  # Wait for full transition before checking game status

            # **Check if the game has ended**
            if check_game_over():
                print("Game over detected! Stopping bot.")
                return True  # Signal that the game is over

            break  # Exit transition loop if game isn't over

    return False  # Game is still active, continue playing
```





1.2 BUG Fixes & Gameplay Optimization!

1.2.1 Sprint Focus: Finalizing the Gameplay Flow

The primary goal of this sprint was to **perfect the gameplay mechanics** of the bot, ensuring smooth and **bug-free** operation in both **Solo** and **Duels Mode**. The **entire gameplay control flow** is now **finalized and optimized**, allowing the bot to play **seamlessly** from start to finish.

With these fixes in place, the remainder of the semester can now be dedicated **solely** to continuously improving the **machine learning model** for better guessing accuracy—without any distractions from gameplay issues.

1.2.2 Key Bugs Fixed:

Recalibrated Coordinate Mapping
- Implemented the `select_regions.py` script to accurately **map pixel positions** to **GeoGuessr minimap coordinates**, reducing misalignment issues.

Game Chat Selection Fix
- Ensured that the bot **unselects game chat** between rounds to prevent it from **blocking minimap screenshots** or corrupting the model's guesses.

Fixed Round Skipping Bug
- Resolved an issue where the bot **made a guess on a blank screen** due to a premature transition between rounds.

Refined End Screen Detection
- Improved **end-game detection** by refining the screenshot capture process to ensure **consistency**

across different Duels matches, reducing false positives.

1.2.3 Impact of These Fixes

- **Smoother gameplay flow** with **no interruptions** in Solo or Duels mode.
- **More accurate guesses** due to **proper coordinate mapping** and **uninterrupted screenshots**.
- **Game-end detection is now more reliable**, ensuring the bot stops at the right time.

With **all gameplay-related bugs resolved**, the next phase is **fully focused on enhancing the AI's prediction capabilities** for **even smarter guesses!**

[ ]:
```python
import pyautogui
from pynput import keyboard
import yaml

# Define the screen and map regions
regions = [
    "screen_top_left",
    "screen_bot_right",
]

map_regions = [
    "map_top_left",
    "map_bottom_right",
    "confirm_button",
    "kodiak",
    "hobart",
]

next_round_button = "next_round_button"
coords = []

# Key to press for capturing coordinates
PRESS_KEY = 'a'

# Function to handle key press
def on_press(key):
    try:
        if key.char == PRESS_KEY:
            x, y = pyautogui.position()
            print(f"Captured coordinates: ({x}, {y})")
            coords.append([x, y])
            return False  # Stop the listener after capturing
    except AttributeError:
        pass
```

```
# Function to get coordinates from user
def get_coords(players=1):
    # Capture screen regions (top left and bottom right)
    for region in regions:
        print(f"Move the mouse to the {region} region and press '{PRESS_KEY}'.")
        with keyboard.Listener(on_press=on_press) as keyboard_listener:
            keyboard_listener.join(timeout=40)

    # Capture map regions for each player
    for p in range(1, players + 1):
        print(f"\nCapturing map regions for player {p}:")
        for region in map_regions:
            specific_region = f"{region}_{p}"
            print(f"Move the mouse to the {specific_region} and press␣
 ↪'{PRESS_KEY}'.")
            with keyboard.Listener(on_press=on_press) as keyboard_listener:
                keyboard_listener.join(timeout=40)
            regions.append(specific_region)

    # Capture the "Next Round" button
    print(f"Move the mouse to the {next_round_button} and press '{PRESS_KEY}'.")
    with keyboard.Listener(on_press=on_press) as keyboard_listener:
        keyboard_listener.join(timeout=40)
    regions.append(next_round_button)

    # Create a dictionary of region names and their coordinates
    screen_regions = {reg: coord for reg, coord in zip(regions, coords)}

    # Save the dictionary as a YAML file
    try:
        with open("screen_regions.yaml", "w") as f:
            yaml.dump(screen_regions, f)
        print("\nCoordinates saved successfully to 'screen_regions.yaml'.")
    except Exception as e:
        print(f"Error saving coordinates: {e}")

    return screen_regions

if __name__ == "__main__":
    # Run the script with one player by default
    _ = get_coords(players=1)
```

[ ]:
```
import pyautogui
from PIL import Image

def capture_end_screen():
```

```
    """
    Captures the bottom quarter of the screen, excluding the last 100 pixels,
    and saves it as a reference image.
    """
    screen_width, screen_height = pyautogui.size()

    # Define the region: bottom quarter, but 100 pixels up excluded
    top = screen_height - (screen_height // 4)  # Bottom quarter start
    height = (screen_height // 4) - 100  # Remove 100 pixels from the bottom

    region = (0, top, screen_width, height)

    # Take a screenshot of the defined region
    screenshot = pyautogui.screenshot(region=region)

    # Save the screenshot
    screenshot.save("end_screen_reference.png")
    print("End screen reference saved as 'end_screen_reference.png'")

if __name__ == "__main__":
    capture_end_screen()
```

[ ]:
```
if "screen_regions.yaml" not in os.listdir():
        screen_regions = get_coords(players=1)
    else:
        with open("screen_regions.yaml") as f:
            screen_regions = yaml.safe_load(f)

    bot = GeoBot(screen_regions, player=1)

    turn = 0
    while True:  # Run indefinitely until the game ends
        turn += 1
        print(f"\nTurn {turn}")

        game_over = play_turn(bot, plot=plot)

        if game_over:
            break  # Stop playing if the game has ended
```

1.3 Dataset Overview (Same as Before)

We are using a dataset of ~50,000 Google Street View images organized into folders by country. The dataset was split into: - **Training set**: 70% of the images, used to train the model. - **Validation set**: 15% of the images, used to tune hyperparameters and evaluate the model during training. - **Test set**: 15% of the images, used to evaluate the final performance of the trained model.

The images were preprocessed to ensure consistent dimensions and normalization.

[ ]:
```
# Data transformations with augmentation
transform = transforms.Compose([
    transforms.Resize(IMAGE_SIZE),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.
 ↪1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Load dataset
dataset = datasets.ImageFolder(DATA_DIR, transform=transform)
class_names = dataset.classes

# Split dataset into train, val, and test
train_size = int(0.7 * len(dataset))
val_size = int(0.15 * len(dataset))
test_size = len(dataset) - train_size - val_size

print("Splitting dataset...")
train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size,␣
 ↪val_size, test_size])

# Data loaders
print("Creating data loaders...")
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE)
```

1.4 Model Architecture: EfficientNet_V2_S

EfficientNet_V2_S is a modern convolutional neural network (CNN) optimized for both accuracy and efficiency. It is an improved version of EfficientNet, developed by Google. It provides better accuracy with fewer parameters, faster training, and a smaller model size, making it more efficient.

[ ]:
```
print("Initializing model...")
model = efficientnet_v2_s(weights="IMAGENET1K_V1")
# Replace the classifier layer to match the number of classes
model.classifier[1] = nn.Linear(model.classifier[1].in_features,␣
 ↪len(class_names))
model = model.to(DEVICE)
```

1.5 Training the Model

The model is implemented into the standard deep learning process.

- **Forward Pass:** Images are passed through EfficientNetV2-S to get predictions.

- **Loss Calculation:** Measures how far predictions are from true labels.
- **Backward Pass:** Computes gradients for optimization.
- **Weight Updates:** `optimizer.step()` updates the model based on gradients.

[ ]:
```
def train_one_epoch(epoch):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for inputs, labels in tqdm(train_loader, desc=f"Training Epoch {epoch + 1}/
 ↪{EPOCHS}"):
        inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Update metrics
        running_loss += loss.item() * inputs.size(0)
        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)

    return running_loss / len(train_loader.dataset), 100 * correct /␣
 ↪len(train_loader.dataset)
```

1.5.1 Wait for Transition (Still Used)

Instead of the user being able to control when to go to the next round, a **Duels** mode round ends when both players have made a guess. Then a comparison, between round screen shows the round result, and a countdown starts before moving on to the next round.

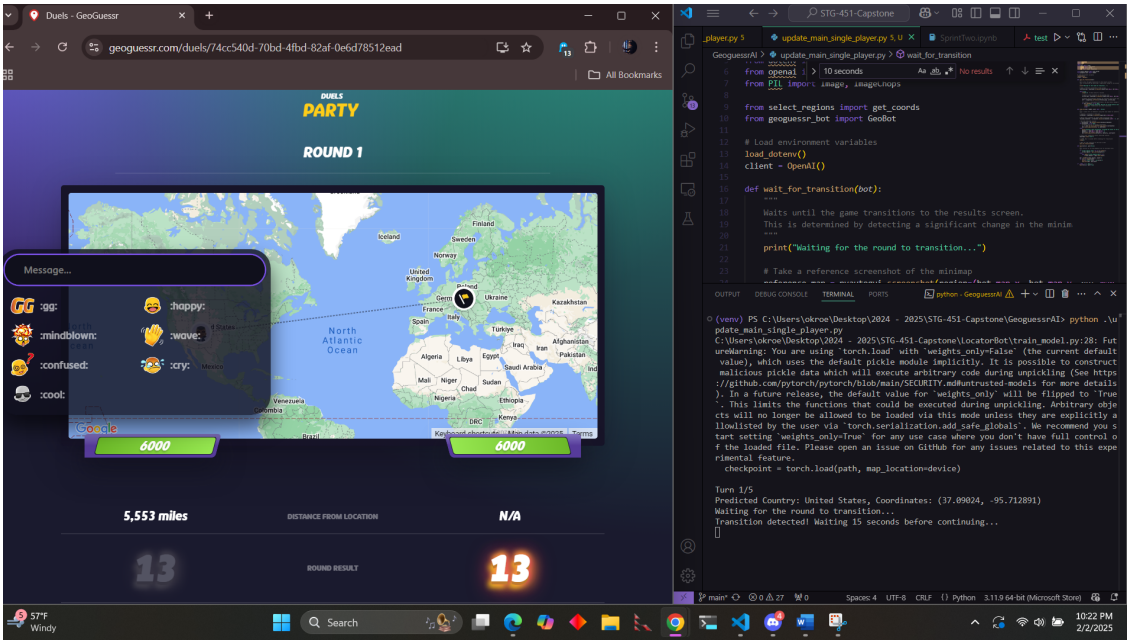
The bot used to be able to click next when it made a guess, but must now wait until the round is over.

A `wait_for_transition` function was implemented that tracks the minimap in the bottom left. Once this map has disappeared / changed, it knows that the round has ended, and it waits the 15 seconds until the next round starts.

[ ]:
```
def wait_for_transition(bot):
    """
    Waits until the game transitions to the results screen.
    This is determined by detecting a significant change in the minimap area.
    """
```

```
    print("Waiting for the round to transition...")

    # Take a reference screenshot of the minimap
    reference_map = pyautogui.screenshot(region=(bot.map_x, bot.map_y, bot.
↪map_w, bot.map_h))

    while True:
        sleep(0.5)  # Check every 0.5 seconds

        # Capture a new screenshot of the minimap area
        current_map = pyautogui.screenshot(region=(bot.map_x, bot.map_y, bot.
↪map_w, bot.map_h))

        # Compute the difference between the reference and current image
        diff = ImageChops.difference(reference_map, current_map)

        # If there's a noticeable difference, the map likely changed␣
↪(transition detected)
        if diff.getbbox() is not None:
            print("Transition detected! Waiting 15 seconds before continuing...
↪")
            sleep(15)  # Additional wait after transition
            break
```


```
Turn 2/5
Predicted Country: Germany, Coordinates: (51.165691, 10.451526)
Waiting for the round to transition...
```



1.6 Challenges

- A few bugs were hard to find, but no real challenges now.

1.7 Next Steps

- Focus on improving model to 85%+ accuracy
- Train on powerful computer (LAB?)
- Find more data and improve regional guessing