Comprehensive Report on RuneScape Botting Project

1. Introduction

This report provides a comprehensive overview of potential APIs, libraries, frameworks, toolkits, and extensions for developing a RuneScape bot, with a focus on both Old School RuneScape (OSRS) and 2004scape (Lost City). It also addresses the feasibility of creating a custom API for color botting, the application of machine learning for in-game image recognition, and how university resources can be leveraged. The report concludes with a detailed 12-week project plan.

2. Botting Methodologies

Botting in RuneScape generally falls into two main categories: client-based botting and pixel/color-based botting. Understanding these methodologies is crucial for selecting appropriate tools and strategies.

2.1. Client-Based Botting

Client-based botting involves directly interacting with the game client's memory or processes. This method typically uses APIs provided by third-party botting clients or involves reverse-engineering the game client itself to read game state directly from memory, inject commands, or reflect game objects. This allows for precise control and access to detailed game information (e.g., player stats, inventory contents, NPC health, object IDs) without relying on visual cues.

- **Advantages:** Highly efficient, precise, and less prone to breaking due to graphical updates. Can perform complex tasks and react quickly to in-game events.
- **Disadvantages:** More susceptible to detection by anti-cheat systems that monitor client modifications or memory access. Requires deep understanding of the game client's internal structure.
- **Applicability:** Primarily for OSRS, where established botting clients provide APIs for this type of interaction. Unlikely to be feasible for 2004scape unless you have access to its client source code or are willing to reverse-engineer it.

2.2. Pixel/Color-Based Botting

Pixel/color-based botting, also known as image recognition botting, operates by analyzing the pixels on the game screen to determine the game state and then simulating human input (mouse clicks, keyboard presses) based on that analysis. This method does not directly interact with the game client's internal processes.

- **Advantages:** Generally less detectable by client-side anti-cheat systems as it mimics human interaction with the screen. Can be used on any game where visual information is available.
- **Disadvantages:** Highly susceptible to breaking due to graphical updates, changes in screen resolution, UI scaling, or even lighting conditions. Can be slower and less precise than client-based methods. Requires robust image processing and recognition algorithms.
- **Applicability:** Ideal for 2004scape where no official client-side APIs exist. Also used for OSRS, particularly by platforms like WASP Scripts, for its lower detection risk compared to direct client manipulation.

3. Potential APIs for Different Bot Types

3.1. APIs for Old School RuneScape (OSRS)

For OSRS, the most effective and commonly used APIs are those provided by third-party botting clients. These clients have already handled the complex task of interacting with the game client, providing a higher-level abstraction for script development.

- **RuneLite API (Java-based):**

    - **Type:** Client-side API. RuneLite is a popular open-source third-party client for OSRS. Its API allows developers to create plugins that run within the RuneLite client itself. This provides access to game data (player position, inventory, NPC data), game events, and methods for simulating actions. It's often considered a "white-hat" approach as it operates within a widely accepted client [1].
    - **Use Case:** Ideal if you want to develop highly integrated bots that leverage RuneLite's features and potentially benefit from its anti-ban measures. Requires Java programming.

- **DreamBot API (Java-based):**

    - **Type:** Client-side API. DreamBot is a well-established OSRS botting client that offers a comprehensive API for script development. It handles many low-level

interactions and often includes built-in humanization and anti-ban features [2].
    - **Use Case:** Suitable for developing robust and feature-rich OSRS bots. Scripts are written in Java using DreamBot's API. DreamBot has a large community and a marketplace for scripts.

- **OSBot API (Java-based):**

    - **Type:** Client-side API. Similar to DreamBot, OSBot is another widely used OSRS botting client with its own API for creating automated scripts. It aims for user-friendliness and provides tools for script development and deployment [3].
    - **Use Case:** Another strong option for OSRS bot development, particularly if you prefer its community or specific features. Scripts are written in Java using the OSBot API.

- **Tribot API (Java-based):**

    - **Type:** Client-side API. Tribot is a long-standing OSRS botting client known for its advanced humanization features and active development. It provides a Java API for script creation [4].
    - **Use Case:** A good choice if human-like behavior and ban avoidance are top priorities.

3.2. APIs for 2004scape (Lost City)

As discussed in previous reports, there are no known dedicated, publicly documented APIs specifically for 2004scape. This means you will not be able to call functions from a game-specific library to get player coordinates or interact with game objects directly. Your "API" for 2004scape will be a combination of lower-level tools.

- **Simulated API through Computer Vision and Input:** Your interaction with 2004scape will primarily be through simulating human input based on visual analysis. This effectively creates a custom, high-level API for your bot:
    - **Visual Input:** Using libraries like OpenCV to visually interpret the game state (e.g., "is a tree there?", "is my inventory full?").
    - **Simulated Output:** Using libraries like PyAutoGUI (Python) or Java Robot class (Java) to send commands to the game (e.g., "click on the tree", "move mouse to bank").

4. Viability of Creating Your Own API

Creating your own API for botting is a significant undertaking, and its viability depends heavily on the target game and your project goals.

4.1. For OSRS

- **Not Recommended (for a 12-week project):** Creating a full-fledged API for OSRS from scratch would involve reverse-engineering the game client, understanding its memory structure, and implementing complex injection or reflection techniques. This is an extremely challenging task that would likely take far longer than your 12-week project timeframe. It also carries a very high risk of detection and bans, as Jagex actively combats such direct client manipulation.
- **Leverage Existing APIs:** For OSRS, it is highly recommended to leverage the APIs provided by established botting clients (RuneLite, DreamBot, OSBot, Tribot). These platforms have already done the heavy lifting of client interaction and provide a more stable and feature-rich environment for script development. Your project can then focus on developing intelligent bot logic and data analysis, rather than low-level client interaction.

4.2. For 2004scape (Lost City)

- **Necessary (in a sense):** Since there are no existing dedicated APIs for 2004scape, you will inherently be creating your own "API" by combining computer vision and input simulation libraries. This isn't an API in the traditional sense of interacting with the game's internal code, but rather a set of functions and modules that abstract the process of perceiving the game world and interacting with it.

- **Feasible and Recommended:** This approach is not only viable but necessary for 2004scape. Your "API" will consist of your own Python (or Java) functions that:

    - Take screenshots.
    - Process images to find objects.
    - Determine game state based on visual cues.
    - Simulate mouse and keyboard actions.

  This is a core part of your project and will be a valuable learning experience in computer vision and GUI automation.

5. Libraries to Explore Using

Based on your existing knowledge of Java and Python, and the requirements of both color and client botting, here's a breakdown of libraries, including those you've already listed:

5.1. Python Libraries (Highly Recommended for 2004scape and Data Analysis)

- **OpenCV (`cv2`):**
    - **Purpose:** Essential for computer vision tasks. It provides functions for image loading, manipulation, template matching (finding small images within larger ones), color detection, and object recognition. Crucial for pixel-based bots.
    - **Your Mention:** You listed this, and it's a perfect fit.
- **Tesseract OCR (`pytesseract`):**
    - **Purpose:** An Optical Character Recognition (OCR) engine. Useful for reading text directly from the game screen (e.g., chat messages, item names, numbers in the UI). You'll need to install the Tesseract executable separately.
    - **Your Mention:** You listed this, and it will be very helpful for reading dynamic text.
- **PyAutoGUI:**
    - **Purpose:** For GUI automation. It allows you to control the mouse and keyboard, perform clicks, type text, and take screenshots. It's your primary tool for simulating user input.
    - **Your Mention:** You listed this, and it's fundamental for input simulation.
- **NumPy:**
    - **Purpose:** The fundamental package for numerical computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. OpenCV often returns images as NumPy arrays.
    - **Your Mention:** You listed this, and it's a prerequisite for efficient image and data manipulation when working with OpenCV and Pandas.
- **TensorFlow (or PyTorch):**
    - **Purpose:** A powerful open-source machine learning framework. While potentially overkill for simple bots, it could be used for more advanced

botting, such as training models for more robust object detection, human-like mouse movement generation, or even predicting optimal actions based on game state.
    - **Your Mention:** You listed this. For your initial project, it might be too complex, but it's excellent for future expansion into AI-driven bots.

- **Pillow (PIL Fork):**
    - **Purpose:** A user-friendly image processing library. Useful for basic image manipulation, loading, saving, and taking screenshots. Often used in conjunction with OpenCV for screen capture.
    - **Relevance:** Complements OpenCV for image handling.

- `sqlite3` **(Built-in Python module):**
    - **Purpose:** For interacting with SQLite databases. SQLite is a lightweight, file-based relational database, perfect for storing your bot activity logs and ban data locally.
    - **Relevance:** Crucial for your data collection and analysis goals.

- **Pandas:**
    - **Purpose:** A powerful data manipulation and analysis library. Provides data structures like DataFrames that are perfect for organizing and analyzing your collected bot data.
    - **Relevance:** Essential for processing and cleaning your bot activity logs before analysis.

- **Matplotlib / Seaborn:**
    - **Purpose:** Data visualization libraries. Matplotlib is the foundational plotting library, while Seaborn builds on it to provide a higher-level interface for drawing attractive statistical graphics.
    - **Relevance:** For visualizing bot performance, ban rates, XP curves, and identifying patterns in your collected data.

5.2. Java Libraries (If pursuing OSRS client-based botting)

- **Java AWT Robot Class:**
    - **Purpose:** Part of Java's Abstract Window Toolkit (AWT), this class allows Java code to generate native system input events for test automation, self-running demos, and other applications that need to control the mouse and keyboard.

    - **Relevance:** Your equivalent to PyAutoGUI if you choose to implement pixel-based bots in Java.

- **OpenCV for Java:**
    - **Purpose:** The Java bindings for OpenCV. Provides the same powerful computer vision capabilities as the Python version.
    - **Relevance:** If you decide to do pixel-based botting in Java, this is your go-to for image processing.

- **SQLite JDBC Driver:**
    - **Purpose:** A JDBC (Java Database Connectivity) driver for SQLite, allowing Java applications to connect to and interact with SQLite databases.
    - **Relevance:** For storing bot data if you're working in Java.

6. Frameworks to Explore Using

In the context of botting, "frameworks" often refer to the overarching structure or platform provided by botting clients, or design patterns for your own bot logic.

6.1. Botting Client Frameworks (Primarily OSRS)

These are the most common "frameworks" for OSRS botting, as they provide the underlying structure and APIs for script development:

- **RuneLite Plugin Development Framework:** If you choose the RuneLite route, you'll be working within its plugin development framework. This involves understanding RuneLite's event system, game object models, and rendering pipeline.
- **DreamBot / OSBot / Tribot Scripting Frameworks:** These clients provide their own internal frameworks for writing scripts. They abstract away many low-level details and offer specialized APIs for RuneScape interactions.

6.2. Custom Bot Framework (For 2004scape and general bot design)

For 2004scape, you'll be building your own framework. Consider these design patterns:

- **State Machine:** A highly recommended design pattern for bot logic. Your bot can transition between different states (e.g., `WOODCUTTING`, `BANKING`, `WALKING_TO_LOCATION`, `FIGHTING`) based on game conditions. This makes your bot logic organized, robust, and easier to debug.

- **Modular Design:** Separate your bot into distinct modules:
  - **Game Interaction Layer:** Handles screen capture, image processing, and input simulation.
  - **Decision-Making Layer:** Implements the bot's logic and state transitions.
  - **Data Logging Layer:** Records all relevant bot activity and ban data.
  - **Humanization Layer:** Applies randomized delays, mouse movements, etc.

7. Toolkits and Extensions

These are often higher-level tools or communities that provide resources or specialized functionalities.

7.1. User-Mentioned Toolkits

- **WASP Scripts:**
  - **Category:** Color Botting Platform and Scripts.
  - **Description:** WASP Scripts (waspscripts.com) is a collection of open-source color scripts primarily written for Old School RuneScape. Their website explicitly states they are "100% color, 100% open source" and built on "Simba on top of SRL and WaspLib." They emphasize using advanced image recognition and computer vision techniques like template matching, OCR, edge detection, and color math to build efficient color OSRS bot scripts [5].
  - **SRL-T (Simba RuneScape Library - Torwent):** This is a library built on top of Simba, designed for RuneScape botting. It provides a set of utilities and functions to interact with the game client, primarily through pixel-based recognition and input simulation. Its documentation outlines various modules for utilities (Keyboard, Mouse, Color, Math, Time, etc.), game interaction (Minimap, Inventory, Bank, Combat, Login, etc.), and anti-ban measures [6].
  - **WaspLib:** This is another library that complements Simba and SRL-T, providing additional functionalities for RuneScape botting. It focuses on abstracting common botting tasks and providing tools for script development, including modules for consumable handling, progress reporting, world hopping, and various development tools like `Tool GetObject` and `Tool Map Maker` [7].
  - **Relevance:** This confirms that WASP is a color botting platform, leveraging Simba and its associated libraries (SRL-T, WaspLib). It reinforces the viability of a pixel-based approach for OSRS, and by extension, for 2004scape. Their open-source nature means you could potentially study their code for implementation insights into color botting techniques.

- **SIMBA:**
    - **Category:** Scripting language and automation tool.
    - **Description:** SIMBA (formerly SRL - Simba's RuneScape Library) is a scripting language and IDE primarily used for automating tasks in RuneScape. It's known for its pixel-based image recognition and robust input simulation capabilities. It's often used for older versions of RuneScape or private servers where direct client interaction is not possible or desired [8].
    - **Relevance:** Highly relevant for 2004scape. If you prefer a dedicated scripting language environment over pure Python/Java for pixel-based botting, SIMBA could be an option. It has a strong community and many pre-built scripts.
- **DreamBot:**
    - **Category:** OSRS Botting Client and Framework.
    - **Description:** As mentioned in the API section, DreamBot is a full-fledged OSRS botting client with a Java API, a script marketplace, and a community. It provides a comprehensive environment for developing and running OSRS bots.
    - **Relevance:** A top contender if you decide to pursue client-based botting for OSRS.
- **OSMB:** This name is not a widely recognized RuneScape botting client or framework in the same vein as DreamBot or OSBot. It might be a typo, a very niche tool, or a misremembered name. If you have more context on OSMB, it would be helpful.

7.2. Other Relevant Toolkits/Extensions

- **RuneLite (for OSRS):** While its API was mentioned, RuneLite itself is a powerful toolkit. Its open-source nature means you can study its code for insights into game interaction, and its plugin system is a direct extension point for your botting efforts.
- **IDE Extensions:** For your chosen IDE (PyCharm, VS Code, IntelliJ IDEA), various extensions can enhance your development workflow, such as linters, debuggers, and version control integrations.

8. Difficulty of Creating an API for Only Color Botting

Creating an "API" for only color botting, as discussed previously, is essentially building a set of functions and modules that abstract the process of screen perception and input simulation. The difficulty of this task is moderate to high, depending on the level of sophistication and robustness you aim for.

8.1. Core Components and Their Difficulty:

1. **Screen Capture:** Relatively easy. Libraries like Python's `Pillow` or `mss` (for faster screenshots) can capture the screen or specific windows. Java's `Robot` class can also take screenshots.

2. **Image Processing and Object Recognition (Moderate to High Difficulty):** This is the most challenging part.

    - **Color Detection:** Fairly easy for simple, distinct colors. More complex if colors vary or are part of a gradient.
    - **Template Matching:** Moderate difficulty. OpenCV's `matchTemplate` function is straightforward to use. The challenge lies in creating robust templates that work across different lighting conditions, zoom levels, and minor graphical variations. You'll need a good collection of reference images for each object you want to detect.
    - **Feature Detection (e.g., SIFT, ORB):** Higher difficulty. These methods are more robust to changes in scale, rotation, and perspective than simple template matching, but they are more complex to implement and fine-tune.
    - **Optical Character Recognition (OCR):** Moderate difficulty. Using Tesseract (via `pytesseract` in Python) is relatively easy. The difficulty comes from ensuring accurate recognition of in-game fonts, sizes, and colors, which often requires pre-processing the image or training custom OCR models.

3. **Input Simulation (Easy to Moderate Difficulty):** Libraries like `PyAutoGUI` or `PyDirectInput` (Python) or Java's `Robot` class make sending mouse and keyboard inputs relatively easy. The challenge lies in making these inputs appear human-like (humanization), which involves adding randomized delays, non-linear mouse paths, and slight variations in click positions.

4. **Game State Management (Moderate Difficulty):** Your API will need to maintain a representation of the game's state based on visual information. This involves:

    - Tracking player position (if visually determinable).
    - Monitoring inventory slots.
    - Detecting NPC presence and health.
    - Identifying active UI elements.
    - This requires careful design of data structures and logic to interpret visual cues into meaningful game state variables.

5. **Robustness and Error Handling (High Difficulty):** A production-ready color botting API needs to be extremely robust. This means handling:
    - Unexpected pop-ups or random events.
    - Disconnections or game client crashes.
    - Changes in game window position or size.
    - Graphical glitches or lag.
    - This often involves extensive testing and iterative refinement.

8.2. Overall Assessment:

For your project, creating a basic color botting API for Lost City (2004scape) that can perform simple tasks like Tutorial Island is **definitely achievable within your timeframe**. It will involve significant effort in image processing and humanization. Building a comprehensive and highly robust color botting API that can handle a wide range of complex tasks and is resilient to minor game updates would be a much larger undertaking, potentially spanning several months or even years of dedicated development.

The key is to **start simple** and incrementally add complexity. Your initial goal of Tutorial Island is an excellent scope for demonstrating the feasibility of a color botting API.

9. Machine Learning for In-Game Image Recognition and Activity Playing

Using machine learning (ML), including neural networks and deep learning, for in-game image recognition and activity playing is a cutting-edge approach in botting. It is **absolutely possible** to train such models on your personal computer, provided you have adequate hardware.

9.1. Feasibility of Training ML Models on a Personal Computer:

- **Hardware Requirements:**
    - **GPU (Graphics Processing Unit):** This is the most critical component. Training deep learning models (like Convolutional Neural Networks for image recognition) is computationally intensive and benefits immensely from a powerful GPU (NVIDIA GPUs with CUDA support are preferred for TensorFlow/PyTorch). While you can train on a CPU, it will be significantly slower, potentially taking days or weeks for models that would train in hours on a good GPU.

    - **RAM:** Sufficient RAM (16GB or more) is important, especially when dealing with large datasets of images.
    - **Storage:** Fast SSD storage is beneficial for loading datasets quickly.

- **Software/Frameworks:**

    - **TensorFlow / Keras:** TensorFlow is a comprehensive open-source ML platform. Keras is a high-level API for building and training deep learning models, making TensorFlow much easier to use. Both are excellent choices for image recognition.
    - **PyTorch:** Another popular open-source ML framework, often favored for its flexibility and Pythonic interface. It's a strong alternative to TensorFlow.
    - **Python:** The primary language for both TensorFlow and PyTorch.

- **Data Collection for Training:**

    - **Labeled Data:** The biggest challenge in using ML for image recognition is acquiring a large, diverse, and accurately labeled dataset of in-game images. You would need to capture screenshots of various in-game objects (e.g., different types of trees, rocks, NPCs, UI elements) under different conditions (lighting, angles, zoom levels) and manually label them.
    - **Data Augmentation:** Techniques like rotation, scaling, cropping, and color jittering can be used to artificially expand your dataset and make your models more robust.

9.2. Use Cases for ML in Botting:

- **Robust Object Detection:** Instead of simple template matching, a trained CNN can identify objects even if their appearance changes slightly, or if they are partially obscured. This is far more resilient to game updates or variations.
- **Human-like Mouse Movements:** Reinforcement learning or behavioral cloning can be used to train models that generate more natural, human-like mouse paths based on observed human gameplay.
- **Activity Playing (Decision Making):** More advanced ML models could potentially learn to make decisions based on the current game state, choosing optimal actions (e.g., which resource to gather, when to bank, how to react to a random event) without explicit rule-based programming.

9.3. Recommendation for Your Project:

For your 12-week project, starting with ML for all image recognition and activity playing might be too ambitious. It involves a steep learning curve for ML concepts, significant data collection, and training time. However, you could:

- **Start with traditional OpenCV methods:** Get your basic bot working with template matching and color detection first.
- **Integrate ML as a stretch goal or for specific, challenging recognition tasks:** For example, if a particular object is hard to detect reliably with template matching, you could dedicate time to training a small ML model just for that object. This would demonstrate the capability without overwhelming the project.
- **Focus on data collection for future ML:** Even if you don't train a full ML model now, systematically collecting labeled screenshots of in-game elements will be invaluable if you decide to pursue ML in a future project or a later phase of this one.

10. Leveraging University Remote Desktop Services and Labs

Your university's remote desktop services and labs are excellent resources that you can leverage for your project, especially for computational resources and IP diversity.

10.1. Computational Resources:

- **Access to Powerful Hardware:** University labs often have high-end workstations with powerful CPUs and, crucially, dedicated GPUs. If your personal computer lacks a strong GPU, these lab machines can significantly accelerate the training of any ML models you decide to implement.
- **Pre-installed Software:** Labs might have common development tools, Python, Java, and even ML frameworks (TensorFlow, PyTorch) pre-installed, saving you setup time.
- **Dedicated Environment:** A lab environment can provide a stable and consistent platform for running long-duration bot tests or ML training sessions without impacting your personal computer's performance or availability.

10.2. IP Diversity and Network Considerations:

- **Different IP Addresses:** Each computer in the university lab network will likely have a different external IP address. If you run multiple bots from different lab machines (with permission and within university policy), each bot would appear to originate from a distinct IP. This is a crucial strategy for avoiding IP-based bans, especially on OSRS.

- **Network Bandwidth:** University networks typically have high bandwidth, which can be beneficial for downloading game clients, updates, and large datasets for ML training.
- **Security and Policy: Crucially, you must check and adhere to your university's IT policies regarding the use of their network and computing resources.** Running bots, even for academic research, might fall under specific usage policies. Ensure you have explicit permission from your supervisor and the IT department if you plan to run automated scripts or access external game servers from university machines, especially if it involves multiple IPs or sustained network activity. Unauthorized use could lead to disciplinary action.

10.3. How to Leverage Them:

1. **Consult IT/Supervisor:** Before using lab machines for botting, discuss your plans with your supervisor and, if necessary, the university's IT department. Explain the academic nature of your project and your need for computational resources and IP diversity.
2. **Remote Access:** Utilize remote desktop services (e.g., RDP, VNC, SSH with X forwarding) to access lab machines from your personal computer. This allows you to develop and monitor your bots remotely.
3. **Dedicated Accounts:** If possible, request dedicated accounts or virtual environments on lab machines to ensure your work is isolated and doesn't interfere with other users.
4. **Data Transfer:** Plan how you will transfer data (code, datasets, logs) between your personal computer and the lab machines (e.g., cloud storage, university network drives, Git repositories).

Leveraging these university resources can significantly enhance the capabilities and scope of your project, particularly for the more computationally intensive aspects like ML training and multi-account testing on live servers. Just ensure you do so responsibly and with proper authorization.

11. OSRS Botting Nerd YouTube Channel Analysis

The OSRS Botting Nerd YouTube channel provides a valuable resource for understanding the practicalities, challenges, and successes in the RuneScape botting scene, particularly through interviews with experienced bot developers and providers. While the channel covers various botting topics, several videos offer direct insights into color botting and general anti-ban strategies.

11.1. Key Insights from Interviews and Videos:

- **Constant Arms Race with Jagex:** A recurring theme is the ongoing battle between bot developers and Jagex's anti-cheat systems (Botwatch). Developers constantly adapt to new detection methods, and Jagex updates its systems to catch new botting techniques. This means a bot that works today might be detected tomorrow.
- **Humanization is Paramount:** Developers consistently emphasize the importance of human-like behavior to avoid bans. This includes:
    - **Realistic Mouse Movements:** Avoiding perfectly straight lines, consistent speeds, and precise clicks. Incorporating natural human imperfections like overshoots, corrections, and varying speeds.
    - **Randomized Delays:** Implementing variable delays between actions to break predictable patterns.
    - **Varied Playtime and Breaks:** Bots running 24/7 or for excessively long, uninterrupted periods are easily flagged. Human players take breaks, log out, and have varied session lengths.
    - **Dynamic Decision Making:** Bots that can adapt to unexpected in-game events or changes in the environment are less likely to get stuck or perform unnatural actions.
- **Graphical Updates (for Color Bots):** Color bots are particularly vulnerable to graphical updates. Even minor changes in textures, UI elements, or lighting can break a color bot's recognition logic, requiring constant maintenance and updates.
- **Anti-Botting Measures by Jagex:** Discussions often touch upon Jagex's methods, including:
    - **Heuristic Analysis:** Analyzing player behavior patterns for deviations from human norms.
    - **Client-Side Detection:** Detecting modified clients or injected code.
    - **Machine Learning:** Jagex uses ML to identify bot-like patterns, making it harder for bots to mimic human behavior perfectly.
    - **Random Events:** While less prevalent now, random events were historically used to test if a player was humanly responsive.
- **Account Management:** The need for careful account management, including using fresh accounts, aging accounts, and avoiding suspicious activities (e.g., large gold transfers to main accounts), is often highlighted.

11.2. Successes and Advancements in Botting (General & Color Botting Specific):

- **Advanced Humanization Techniques:** Successful bot developers focus on sophisticated humanization algorithms that go beyond simple randomization. This includes:
  - **Mouse Splines/Curves:** Generating smooth, natural-looking mouse paths instead of linear movements.
  - **Adaptive Behavior:** Bots that can learn from their environment or adapt their strategy based on in-game conditions.
  - **Background Activity:** Simulating background human activity (e.g., occasionally checking stats, moving camera) even when performing a repetitive task.
- **Modular Script Design:** Breaking down bot scripts into modular components makes them easier to develop, debug, and update in response to game changes or anti-cheat measures.
- **Community Collaboration:** The botting community often shares knowledge, techniques, and tools, accelerating development and helping to overcome challenges.
- **Data-Driven Optimization:** Some developers discuss using data collected from bot runs (e.g., ban rates, uptime) to refine their scripts and humanization parameters, leading to more resilient bots.
- **Color Bot Resilience (with advanced techniques):** While vulnerable to graphical updates, advanced color bots that use robust image recognition (e.g., template matching with high tolerance, feature detection, or even ML-based vision) can still be highly effective, especially if they are well-maintained and updated frequently. The WASP Scripts website explicitly states their use of "advanced image recognition and computer vision techniques like template matching, OCR, edge detection and color math" [5].
- **Private Servers as Testing Grounds:** The concept of using private servers (like 2004scape) for initial development and testing is implicitly supported, as they often have less stringent anti-cheat measures, allowing developers to refine core bot logic before attempting to run on official servers.

11.3. Specific Videos of Interest (Examples from the channel):

- **"OSRS Botting Company Owner Interview"**: Likely discusses the business side, challenges of maintaining a bot farm, and anti-ban strategies from a provider's perspective.

- **"OSRS Botting Top Tier Scriptwriter Interview - GP farming + PVM"**: Provides insights into developing complex scripts for profitable activities and Player-versus-Monster (PVM) content.
- **"OSRS DreamBot Vs Wasp Vs Tribot - Most Risky Scripts?"**: This video directly compares different botting clients, including WASP (a color botting platform), and likely discusses their respective ban rates and risks for various script types. This would be highly relevant to your project.
- **"Are OSRS Color Bots Safe? WASP"**: This video specifically addresses the safety and viability of color bots, using WASP as an example. This is a direct answer to your question about color botting and its effectiveness.

12. Conclusion and Recommendations

Your project is ambitious but highly achievable with a structured approach. Here are the key recommendations:

12.1. For 2004scape (Lost City) - Initial Focus

- **Language: Python** is strongly recommended due to its excellent libraries for computer vision and GUI automation.
- **Core Libraries:** Focus on mastering **OpenCV**, **PyAutoGUI** (or PyDirectInput), **Tesseract OCR**, and **NumPy**. These will form the backbone of your pixel-based bot.
- **Data Storage:** Use **SQLite** with Python's `sqlite3` module for local data logging.
- **Data Analysis:** Leverage **Pandas**, **Matplotlib**, and **Seaborn** for analyzing your collected bot data.
- **Framework:** Implement a **State Machine** design pattern for your bot's logic to keep it organized and robust.
- **Custom API:** You will effectively be building your own "API" by combining these libraries to interpret the screen and simulate input. This is the correct approach for 2004scape.
- **SIMBA:** Explore SIMBA as an alternative if you find pure Python pixel-based botting too challenging or if you prefer a more specialized scripting environment for this type of botting.

12.2. For Old School RuneScape (OSRS) - Future Expansion

- **Language:** If you transition to OSRS client-based botting, **Java** will be the primary language, as most established OSRS botting clients (RuneLite, DreamBot, OSBot, Tribot) use Java for their APIs.
- **APIs:** Choose one of the established client APIs (RuneLite, DreamBot, OSBot, Tribot) and dedicate time to understanding its documentation and examples. **Do**

not attempt to build your own client-level API for OSRS within your project timeframe.
- **Humanization:** Regardless of the client/API chosen, focus heavily on implementing advanced humanization techniques, as OSRS has sophisticated anti-cheat systems.

12.3. General Recommendations

- **Start Simple:** Begin with your plan of creating an account and doing Tutorial Island on Lost City. This will allow you to get a basic bot working and understand the challenges of screen interpretation and input simulation.
- **Iterative Development:** Build your bot in small, testable increments. Get one task working reliably before adding more complexity.
- **Data-Driven Approach:** Your commitment to data collection and analysis is excellent. This will be your most powerful tool for understanding bot performance and ban rates.
- **Ethical Considerations:** Continue to adhere to the academic nature of your project. The goal is learning and research, not commercial gain or disruption of game integrity.

13. 12-Week Project Plan

This is a proposed 12-week project plan, broken down into phases, assuming a start date around early July and a deadline of September 5th. This plan is flexible and can be adjusted based on your progress and findings.

Week 1-2: Setup and Basic Botting (Lost City Focus)

- **Objective:** Get a local 2004scape server running, set up your development environment, and create a basic bot for account creation and Tutorial Island.
- **Tasks:**
    - Download and set up the Lost City server locally.
    - Choose your primary programming language (Python recommended for 2004scape).
    - Install necessary libraries: OpenCV, PyAutoGUI, Tesseract OCR, NumPy.
    - Develop a script to automate account creation on your local Lost City server.
    - Develop a basic pixel/color bot script to complete Tutorial Island.
    - Implement basic screen capture and template matching for object recognition.
    - Implement basic mouse and keyboard input simulation.
    - Set up a local SQLite database for initial data logging (e.g., bot start/end times, account names).

- **Deliverables:** Working local Lost City server, basic account creation script, Tutorial Island bot (proof of concept).

Week 3-4: Advanced Pixel Botting & Humanization (Lost City Focus)

- **Objective:** Enhance your pixel bot with more robust image recognition and humanization techniques. Expand bot activities beyond Tutorial Island.
- **Tasks:**
    - Refine image recognition: Experiment with different template matching parameters, consider basic feature detection.
    - Implement advanced humanization: Non-linear mouse movements, randomized delays, slight variations in click positions.
    - Develop scripts for simple in-game activities (e.g., woodcutting, mining, fishing) on Lost City.
    - Improve data logging: Record XP gained, activities performed, session duration.
    - Start designing your bot's state machine for more complex logic.
- **Deliverables:** More robust pixel bot, scripts for 1-2 additional activities, improved data logging.

Week 5-6: Data Analysis & Dashboard (Lost City Focus)

- **Objective:** Develop a basic dashboard to visualize bot activity and collected data. Analyze initial bot performance.
- **Tasks:**
    - Use Pandas to process and analyze collected bot data from SQLite.
    - Create visualizations using Matplotlib/Seaborn for metrics like XP rates, activity duration, and bot uptime.
    - Develop a simple web-based or desktop dashboard to display key metrics (e.g., using Flask for web, or a simple GUI library).
    - Analyze data to identify patterns, potential issues, and areas for bot improvement.
- **Deliverables:** Functional data analysis scripts, basic bot activity dashboard.

Week 7-8: Multi-Botting & Scalability (Lost City Focus)

- **Objective:** Explore running multiple bots concurrently on your local Lost City server. Address scalability challenges.
- **Tasks:**
    - Investigate methods for running multiple instances of your bot (e.g., separate processes, virtual environments).

    - Manage multiple bot accounts and their states.
    - Optimize your bot's resource usage to support multiple instances.
    - Refine your data collection to handle data from multiple bots.
    - Consider basic load balancing or task distribution among bots.
- **Deliverables:** Ability to run multiple bots concurrently, refined multi-bot data collection.

Week 9-10: OSRS Exploration & ML Introduction (Optional/Stretch Goal)

- **Objective:** Begin exploring OSRS botting and, if feasible, introduce basic ML for image recognition.
- **Tasks:**
    - Research a chosen OSRS botting client (e.g., DreamBot, OSBot) and its Java API.
    - Attempt to write a very simple script using the chosen OSRS client API (e.g., basic login, walking).
    - If time permits and hardware allows, begin collecting labeled image data for a specific in-game object (e.g., a tree, a rock) for ML training.
    - Experiment with a simple TensorFlow/Keras model for object detection (e.g., using a pre-trained model for transfer learning or a small custom CNN).
    - Investigate university remote desktop services for computational resources and IP diversity.
- **Deliverables:** Basic OSRS client script (optional), initial ML image dataset, preliminary ML model (optional).

Week 11: Refinement, Documentation & Reporting

- **Objective:** Refine existing bots, finalize data analysis, and prepare comprehensive project documentation.
- **Tasks:**
    - Address any bugs or inefficiencies in your bots.
    - Ensure all data collection is robust and accurate.
    - Write detailed documentation for your code, setup process, and bot logic.
    - Prepare your final project report, summarizing your methodologies, findings, challenges, and future work.
    - Ensure all references are correctly cited.
- **Deliverables:** Polished bots, comprehensive project documentation, draft of final report.

Week 12: Final Presentation & Submission

- **Objective:** Prepare and deliver your final project presentation and submit all deliverables.
- **Tasks:**
    - Create a compelling presentation summarizing your project, key achievements, and insights.
    - Practice your presentation.
    - Submit your code, documentation, data, and final report.
    - Reflect on the project and identify areas for future research or development.
- **Deliverables:** Final project presentation, complete project submission.

14. References

[1] OSRS Wiki. (n.d.). Botting. Retrieved from https://oldschool.runescape.wiki/w/Botting

[2] Reddit. (2024, February 14). Creating a bot for a rsps 2004 version. Retrieved from https://www.reddit.com/r/RunescapeBotting/comments/1ijccxt/creating_a_bot_for_a_rsps_2004_version/

[3] Reddit. (2024, February 14). Creating a Smart Color Bot with Python's FastAPI. Retrieved from https://www.reddit.com/r/RunescapeBotting/comments/1aq8l2h/creating_a_smart_color_bot_with_pythons_fastapi/

[4] Tribot. (n.d.). Tribot Official Website. Retrieved from https://tribot.org/

[5] WaspScripts. (n.d.). Free Open Source OSRS Bots. Retrieved from https://waspscripts.com/

[6] Torwent. (n.d.). SRL-T documentation. Retrieved from https://torwent.github.io/SRL-T/

[7] Torwent. (n.d.). WaspLib documentation. Retrieved from https://torwent.github.io/WaspLib/

[8] Simba. (n.d.). Simba Official Website. Retrieved from https://simba.community/