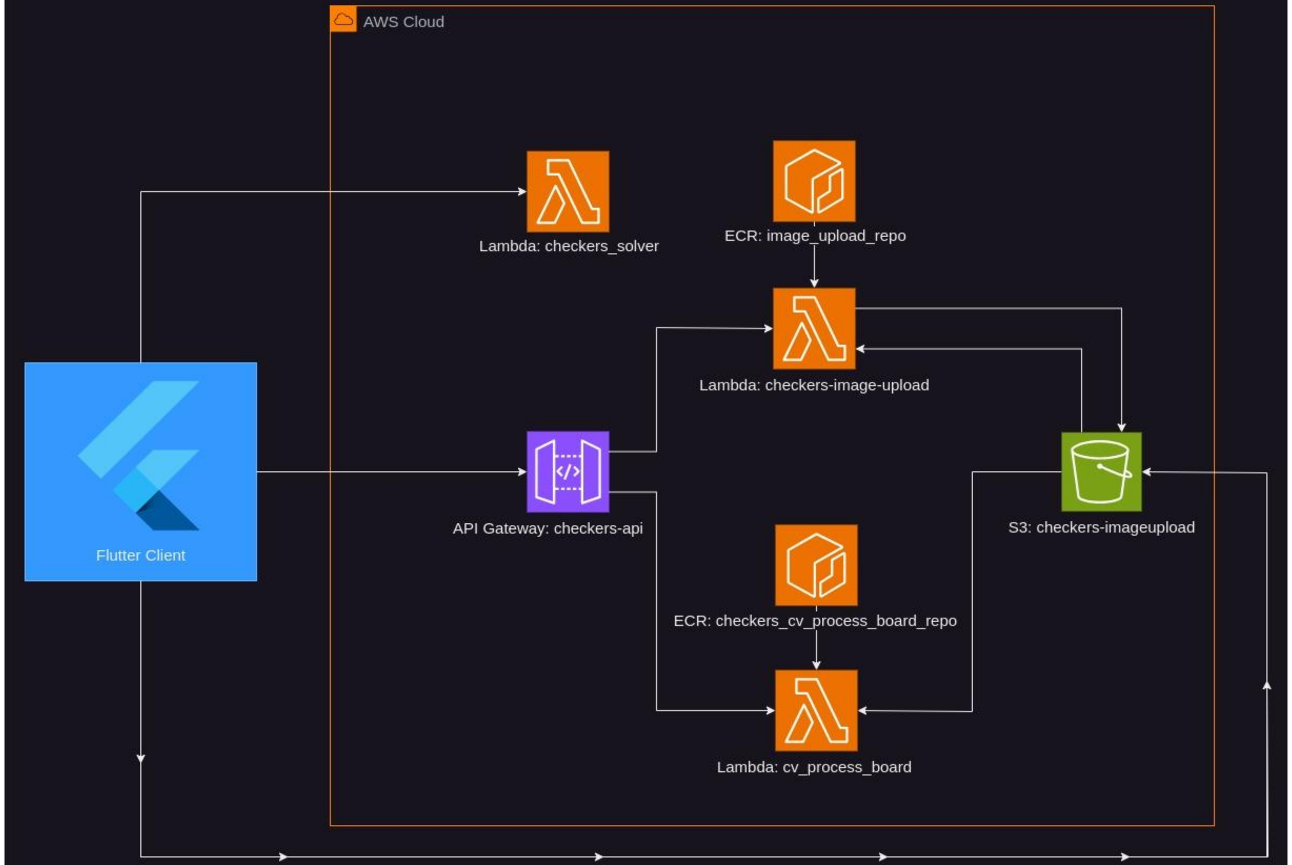
CheckersVision

Charles Dudley
Ranai Srivastav
Sibhat Yosef
Siddhant Jaju
Yaaseen Basha

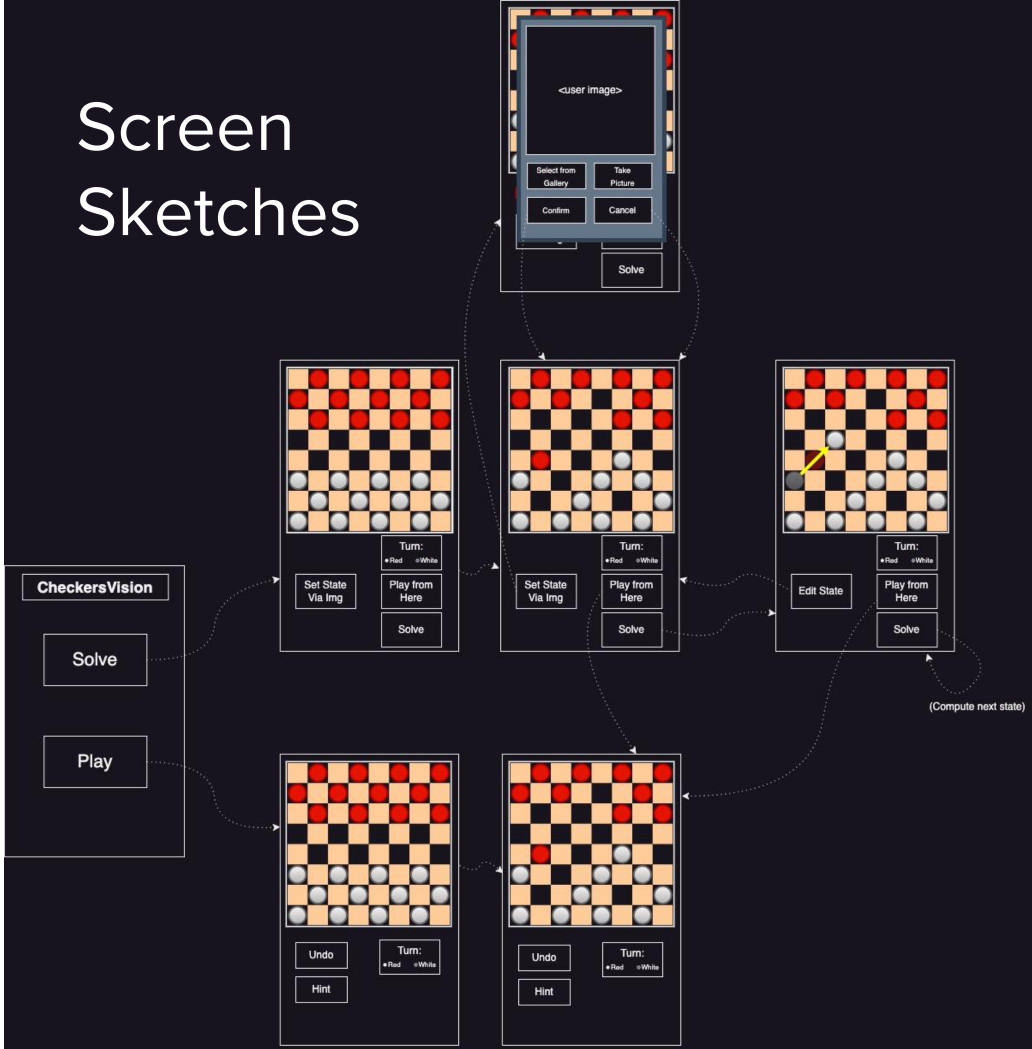
Motivation for CheckersVision

- A cross-platform application where users can:
    - Play a complete Checkers game against an AI
    - Learn what the next best move is based on:
        - A photo of a Checkers board (Computer Vision)
        - Manual input
- A platform for our team to explore various technologies

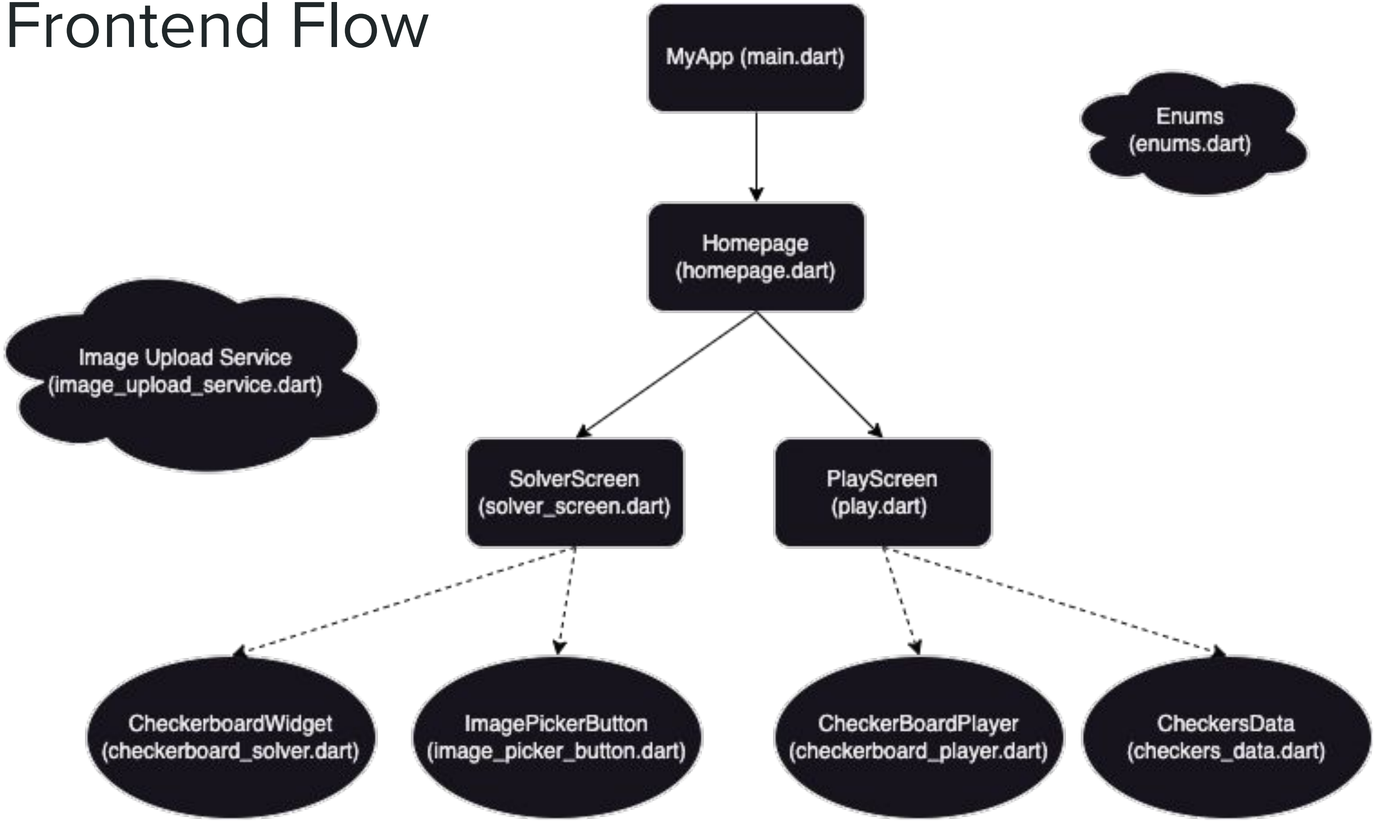
Infrastructure



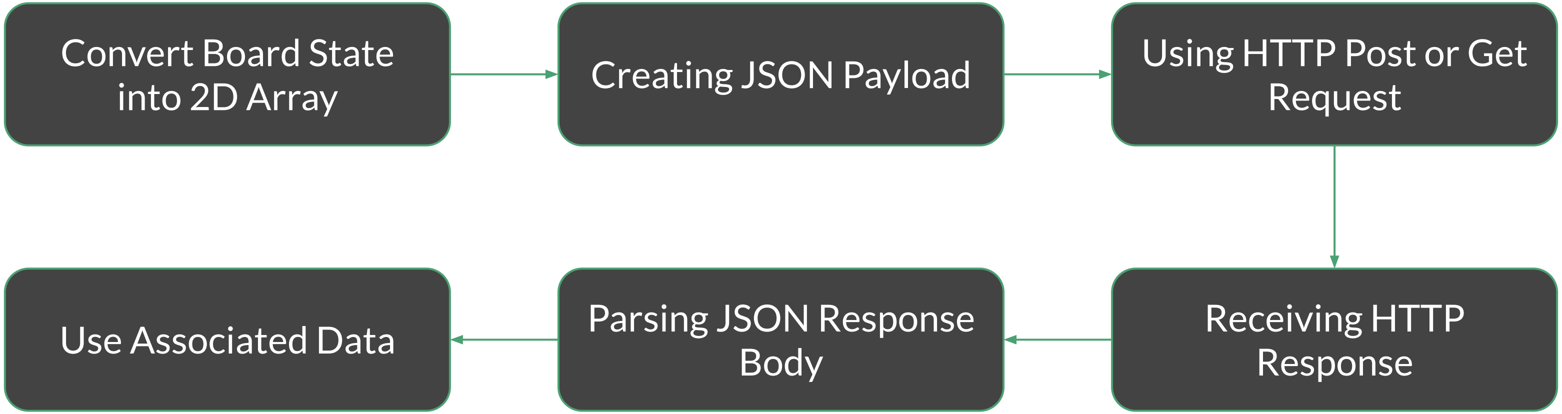
Screen Sketches



Flutter Frontend Flow



Communicating with AWS Backend for Best Move



Movement through screens

```dart
  onPressed: () {
    if (savedBoardState != null) {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => PlayScreen(
            initialState: c_data.CheckersData.fromCustomState(
                savedBoardState!), // c_data.CheckersData.fromCustomState
            isRedTurn: isRedTurn,
          ), // PlayScreen
        ), // MaterialPageRoute
      );
    } else {
      print('Board state is not available.');
    }
  },
  child: Text("Play from Here"),
), // ElevatedButton
// Padding
```

Utilizing state variables (initialState, isRedTurn) to track across screens

```dart
const PlayScreen({Key? key, this.initialState, this.isRedTurn = true})
    : super(key: key);
```

Play Screen Widget Constructor

Frontend - Challenges

- New people working on frontend
- Merge conflicts
- Testing http requests with Chrome

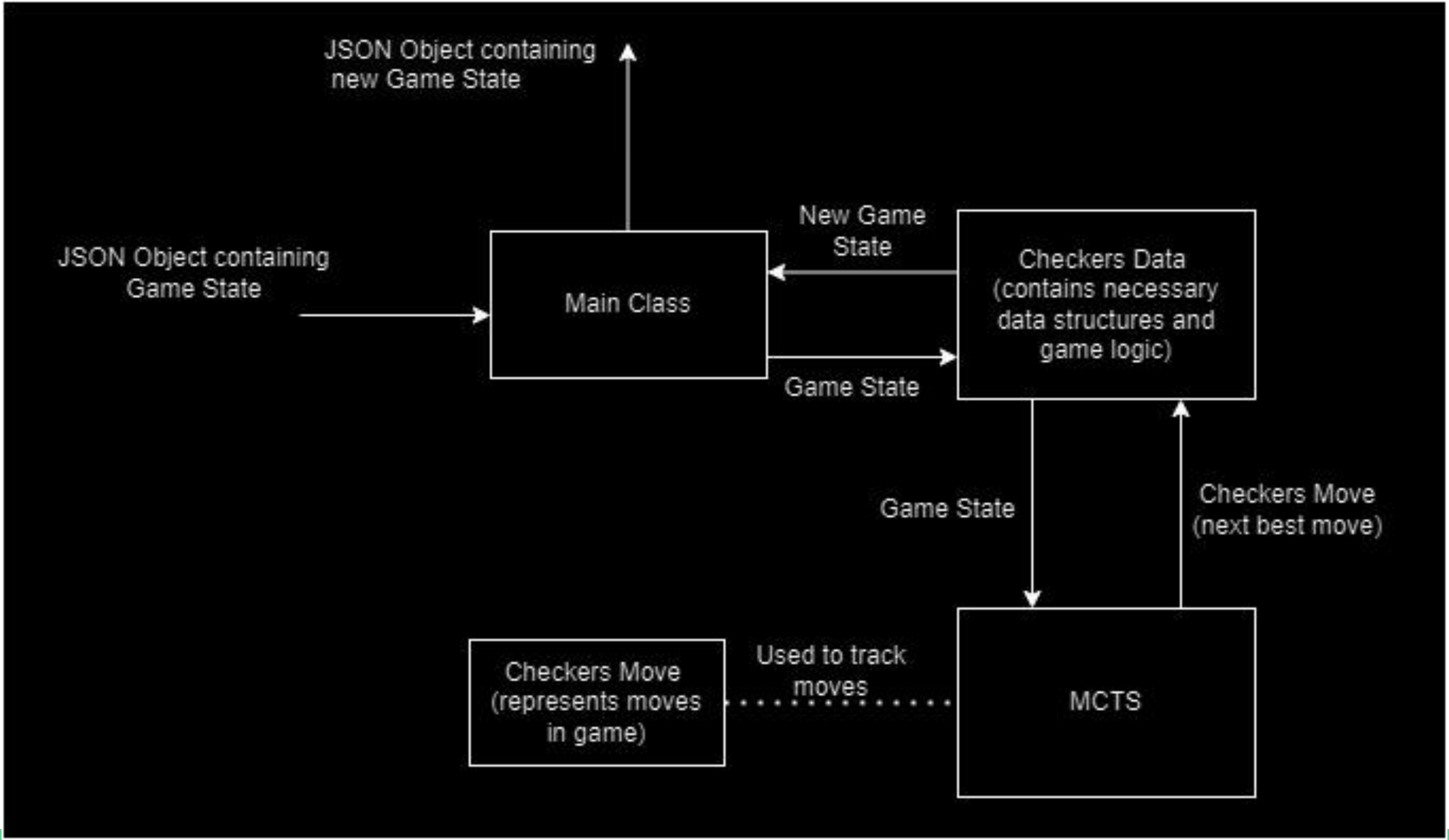
Checkers Engine

What is our Checkers Engine?

- **Backend implementation** - has the **design and logic** for the Checkers Game

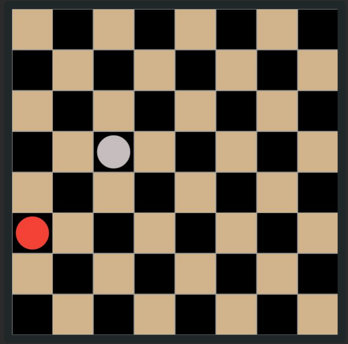
What are the components?

- **Checkers Data**: Contains all the necessary data structures needed for the checkers engine
- **Checkers Move**: Contains the logic of how the pieces will move
- **MCTS:** Main search algorithm used to determine the next best/most optimal move for the user

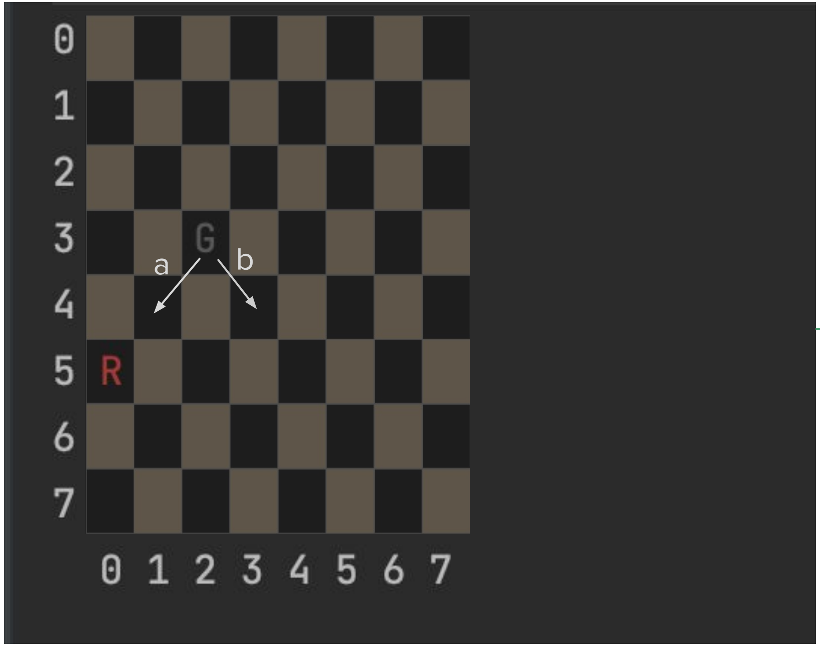

JSON Object containing new Game State
JSON Object containing Game State
Main Class
New Game State
Game State
Checkers Data (contains necessary data structures and game logic)
Game State
Checkers Move (next best move)
Checkers Move (represents moves in game)
Used to track moves
MCTS

Test MCTS for Grey Piece

```java
@Test
public void testMCTSSelectsLegalMoveForSpecificBoard() {
    int[][] board = {
            {0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 1, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0},
            {-1, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0}
    };
    data.setBoard(board);
    System.out.println(data);
    CheckersMove selectedMove = mcts.makeMove(CheckersData.GREY, data);
    data.makeMove(selectedMove);
    assertEquals(CheckersData.GREY, data.pieceAt(4, 3, CheckersData.GREY), "Grey piece should end up at (4,3).");
    System.out.println(data);
}
```

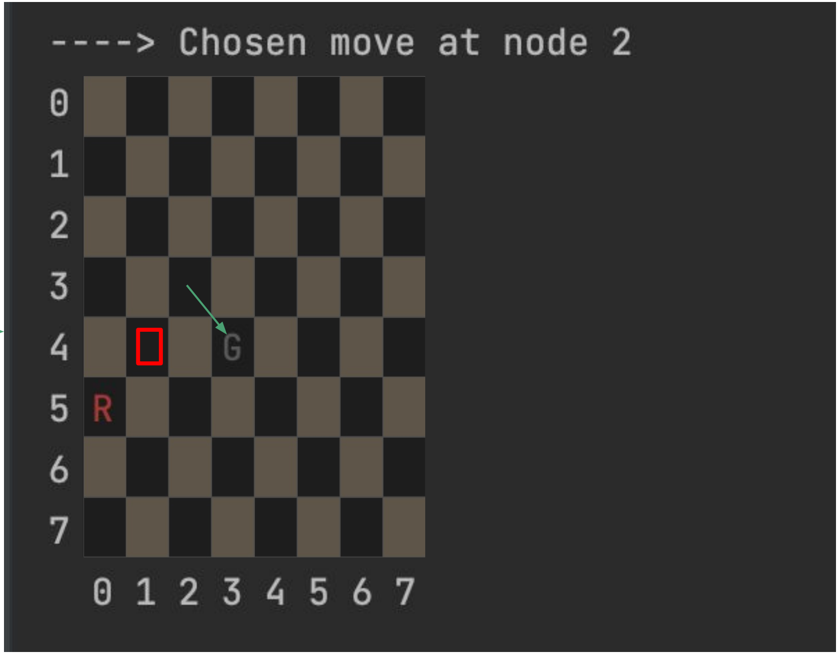


Movement of Grey Piece



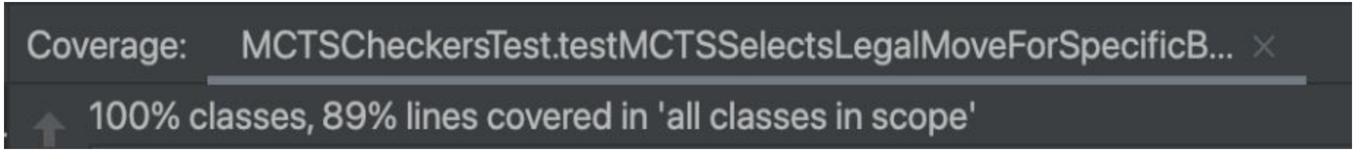
Initial Position (3,2)

MCTS



Final Position (4,3)

Coverage for the Checkers Engine

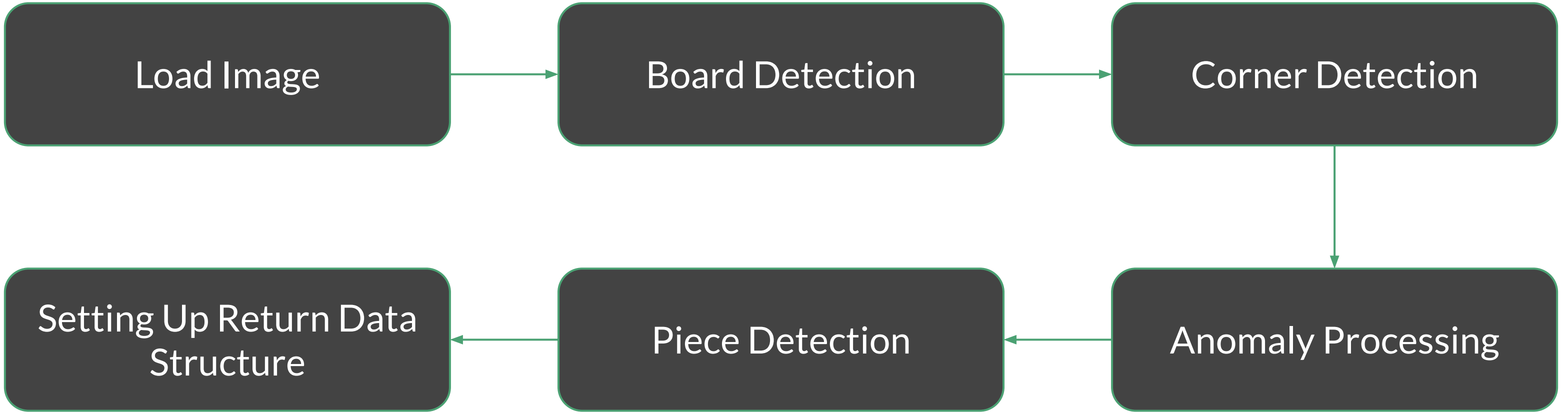


Element	Class, %	Method, %	Line, %
CheckersData	100% (1/1)	90% (20/22)	94% (229/243)
CheckersMove	100% (1/1)	80% (4/5)	93% (14/15)
MCNode	100% (1/1)	100% (15/15)	100% (23/23)
MCTSCheckersTest	100% (1/1)	50% (5/10)	43% (23/53)
MonteCarloTreeSearch	100% (1/1)	100% (11/11)	98% (113/115)

Checkers Engine - Challenges

- Designing the data structures for the Checkers Engine
- Finding total coverage for Checkers Data
- “Lambda-ifying” Checkers Engine

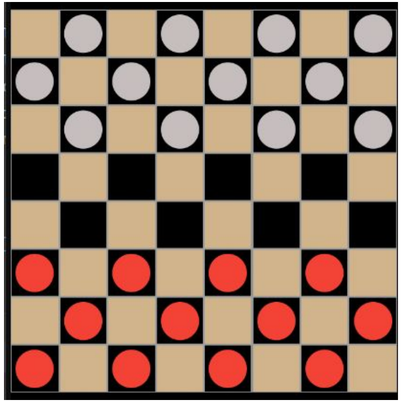
Computer Vision Pipeline with OpenCV



Detecting pieces by what the color is at the midpoint of each square instead of detecting circles

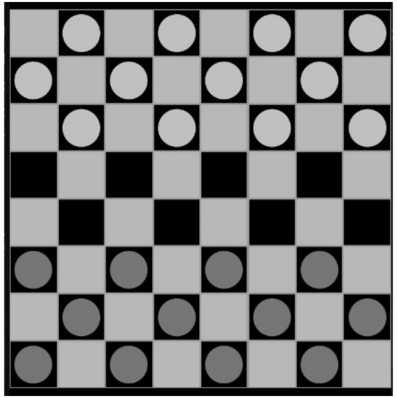
Load Image

We load the image in 3 color spaces



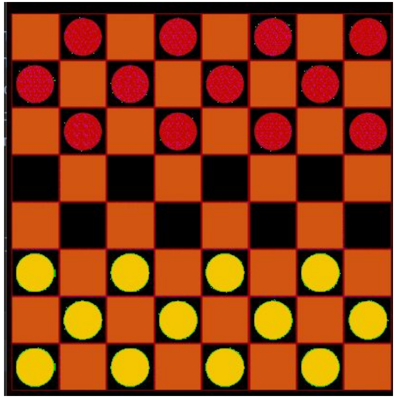
BGR color space

+ Visualization
+ Debugging
- High color sensitivity



Greyscale color space

+ Faster gradient computation for edge detection
- Disregards color



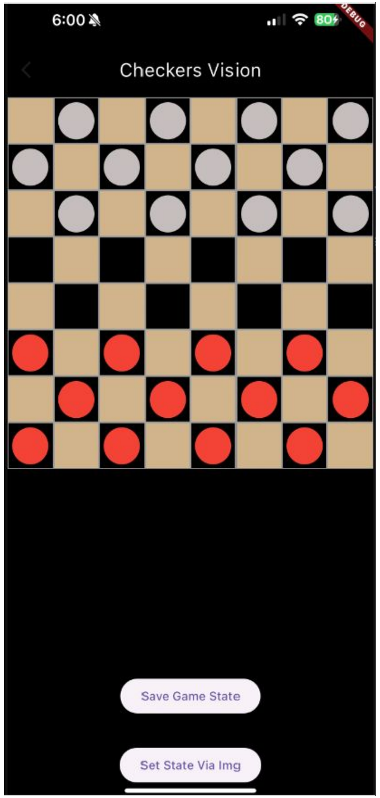
HSV color space

+ Low sensitivity to lighting changes
+ Ideal for color detection
- Hard to visualize

Board Detection

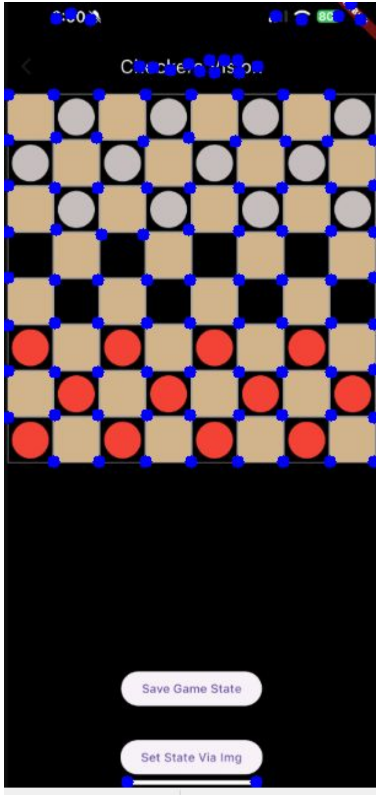
We tested the following approaches for robust edge detection:

- goodFeaturesToTrack
- findChessboardCorners
- Harris Corner Detection
- Hough Line Detection

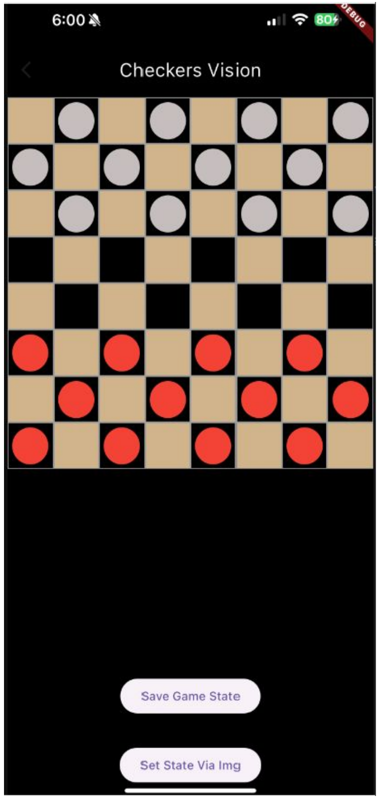


Input image

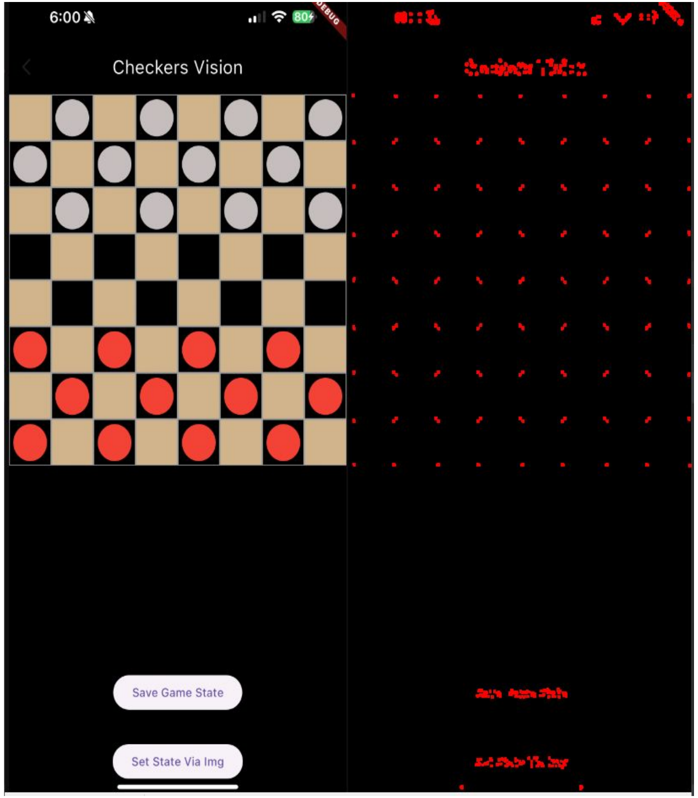
goodFeaturesToTrack



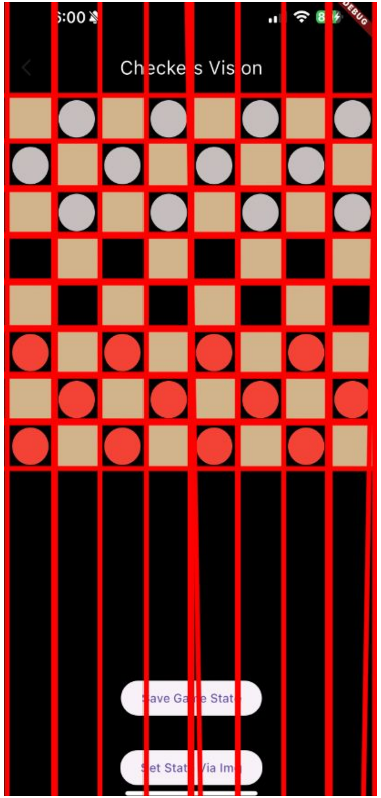
findChessboardCorners



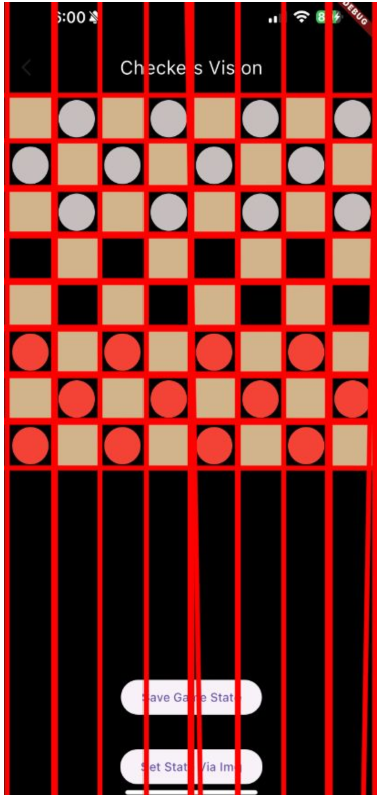
Harris Corner Detection



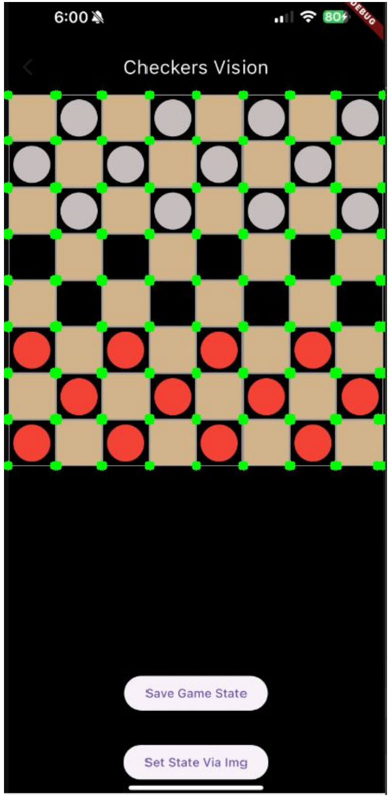
Hough Line Detection

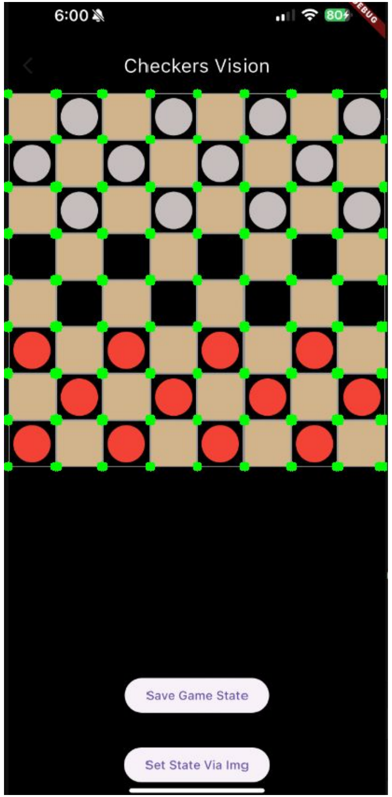


Corner Detection



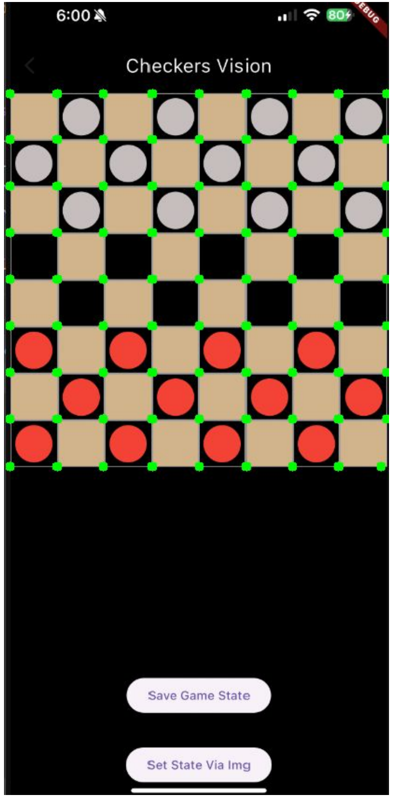
Find corners using intersection of lines





Anomaly Processing

- Remove duplicate corners by removing points that seem too close to one another (5 pixels)
- Ensures 8 rows with 8 columns for color values
- Color values are in range and return UNKNOWN for the square if not in range



Piece Detection

With corners, create squares

```python
class Square:
    def __init__(self, top_left, bottom_left, top_right, bottom_right) -> None:
        self.top_lft = top_left
        self.bot_lft = bottom_left
        self.top_rt = top_right
        self.bot_rt = bottom_right

    def get_midpoint(self):
        x = (self.top_rt[0] + self.top_lft[0]) // 2
        y = (self.bot_lft[1] + self.top_lft[1]) // 2
        return x, y
```

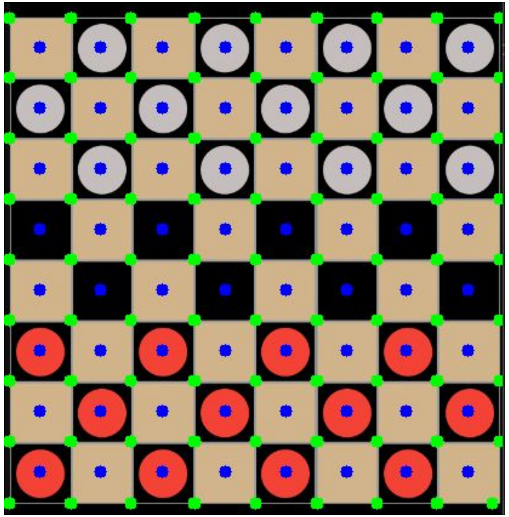
Piece Detection

Sort corners by y-coordinate to group rows

Then sort by x-coordinate to get board

```
Row 1 -: [(4, 101), (54, 101), (103, 101), (155, 101), (205, 101), (255, 101), (304, 101), (353, 101), (404, 101)]
Row 2 -: [(4, 150), (54, 150), (103, 150), (155, 150), (205, 150), (255, 150), (304, 150), (353, 150), (404, 150)]
Row 3 -: [(4, 200), (54, 200), (103, 200), (155, 200), (205, 200), (255, 200), (304, 200), (353, 200), (404, 200)]
Row 4 -: [(4, 250), (54, 250), (103, 250), (155, 250), (205, 250), (255, 250), (304, 250), (353, 250), (404, 250)]
Row 5 -: [(4, 300), (54, 300), (103, 300), (155, 300), (205, 300), (255, 300), (304, 300), (353, 300), (404, 300)]
Row 6 -: [(4, 350), (54, 350), (103, 350), (155, 350), (205, 350), (255, 350), (304, 350), (353, 350), (404, 350)]
Row 7 -: [(4, 400), (54, 400), (103, 400), (155, 400), (205, 400), (255, 400), (304, 400), (353, 400), (404, 400)]
Row 8 -: [(4, 450), (54, 450), (103, 450), (155, 450), (205, 450), (255, 450), (304, 450), (353, 450), (404, 450)]
Row 9 -: [(4, 501), (54, 501), (103, 501), (155, 501), (205, 501), (255, 501), (304, 501), (353, 501), (398, 501)]
```

Kinged Piece



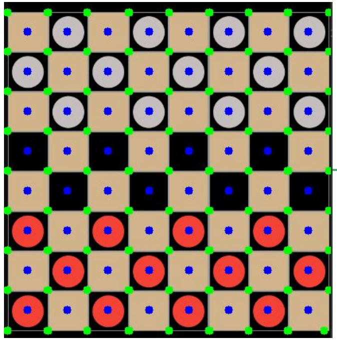
Piece Detection

- Hough circles vs **<u>Color at midpoint</u>**
- Kinged piece detection

```
Square 1 - HSV Values: [ 18  87 209], Color: Empty Peach
Square 2 - HSV Values: [  0   9 197], Color: White
Square 3 - HSV Values: [ 17  85 209], Color: Empty Peach
Square 4 - HSV Values: [  0  12 198], Color: White
Square 5 - HSV Values: [ 17  85 209], Color: Empty Peach
Square 6 - HSV Values: [  0  12 198], Color: White
Square 7 - HSV Values: [ 17  86 210], Color: Empty Peach
Square 8 - HSV Values: [  0  10 197], Color: White
Square 9 - HSV Values: [  0   9 197], Color: White
```

Setting Up Return Data Structure

With color of pieces identified, create board representation to send it back to the frontend



-2 is RED King
-1 is RED Normal
0 is EMPTY Square
+1 is GREY Normal
+2 is GREY King

0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-1	0	-1	0	-1	0	-1	0
0	-1	0	-1	0	-1	0	-1
-1	0	-1	0	-1	0	-1	0

CV - Challenges and limitations

Challenges

- Handling **duplicate** corner detections
- Ignoring **non-board** items
- Piece color detection
- Implementation of identifying **kinged pieces**

OpenCV for image processing

Limitations

- Boards with **tan and white checkers** with **red and grey pieces**
- Picture cannot be rotated **(Camera axis = Board axis)**
- Picture should not have occlusions or shadows

AWS Architecture

- **Serverless** architecture hosted on AWS
- **AWS Services used**
    - S3 (Simple Storage Service)
    - Lambda
    - API Gateway
    - ECR (Elastic Container Registry)
    - CloudWatch

API Gateway

Lambda

S3

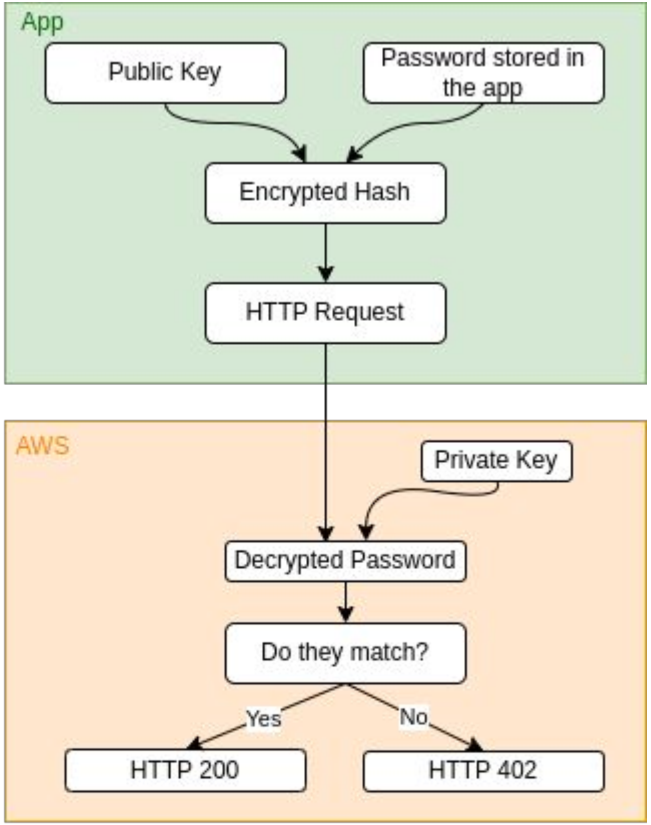
ECR

CloudWatch

Authenticating Lambda function calls

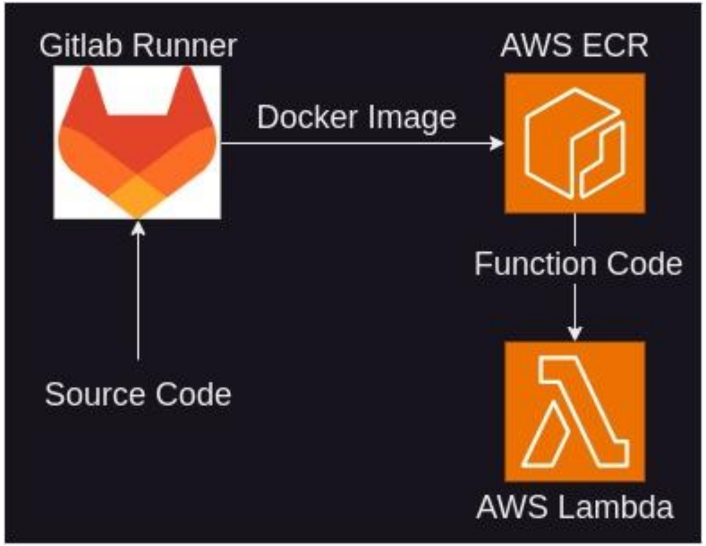
- Prevents unauthorized usage of
  - Our Lambda functions
    - CV
    - Pre-signed URL generation
    - Checkers Engine
  - S3 Bucket

- Attempted ideas
  - Public-private key encryption
  - JWT Token authentication
  - Asymmetric encrypted JWT Authentication



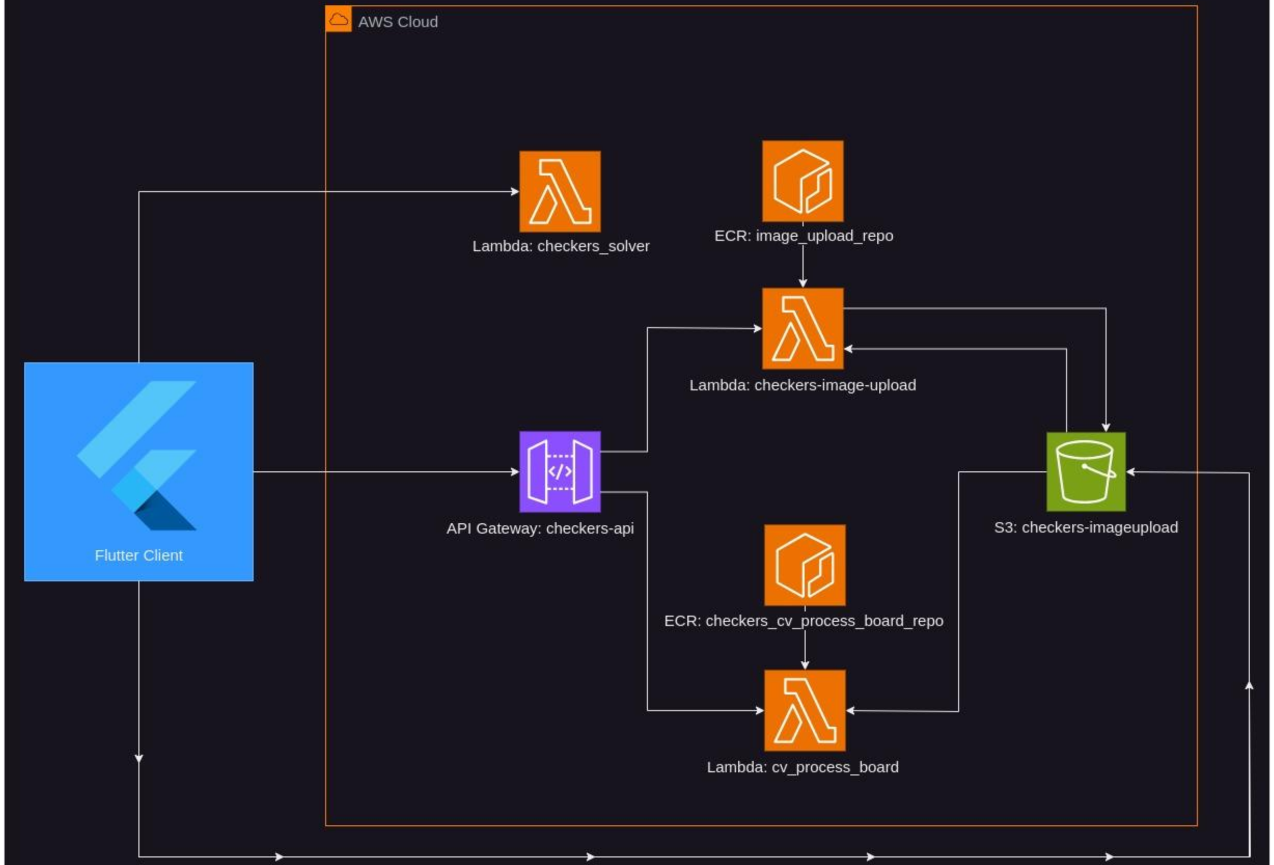
CI/CD with Gitlab Runners

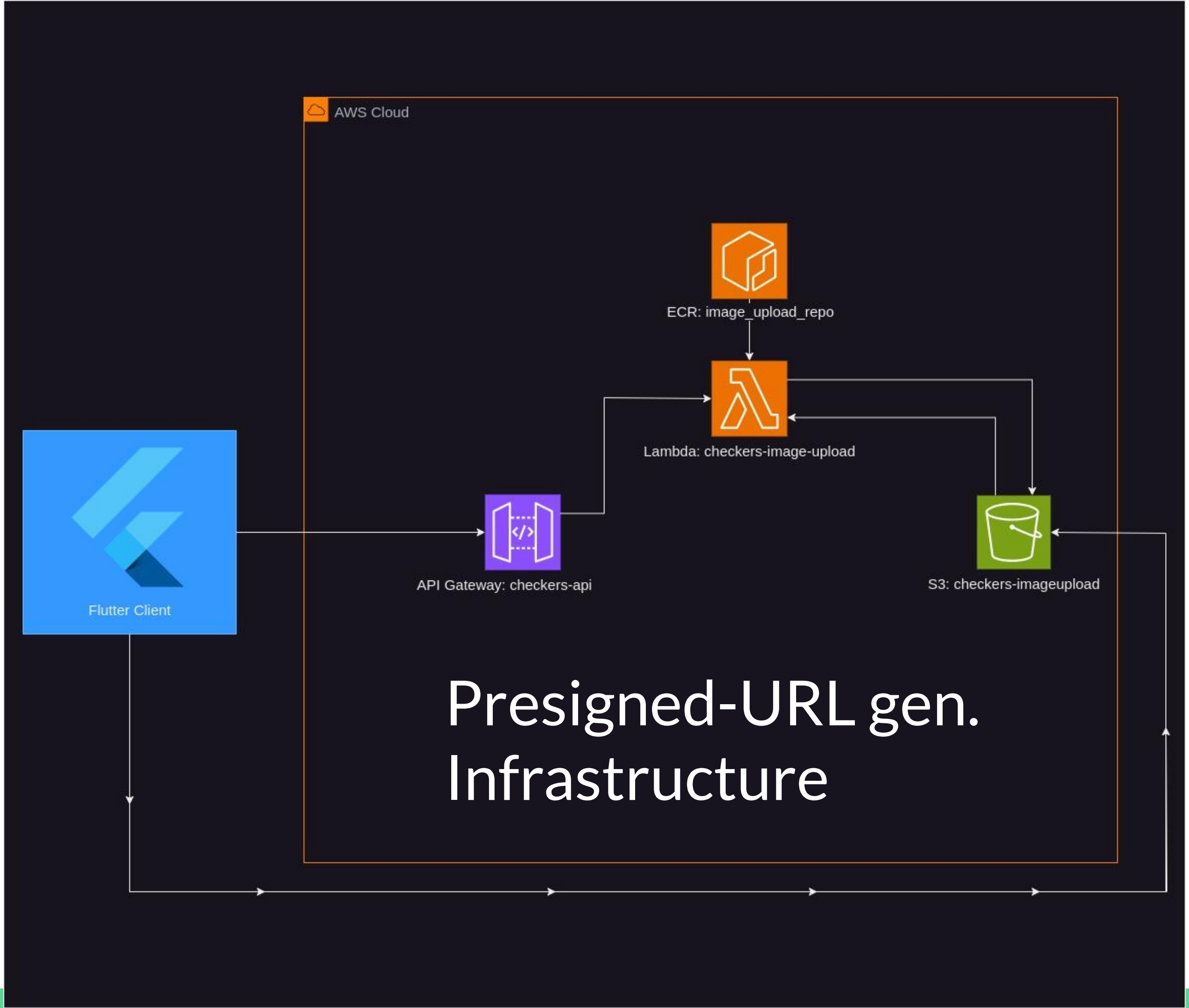
- **Lambdas pull function code from ECR***
  - Package dependencies
  - Facilitate more efficient source control & development
- **What our Runners do:**
  - Clone our project (at most recent commit)
  - Build docker image
  - Tag image as “latest”
  - Push image to AWS ECR
  - Update Lambdas to pull from new “latest” image



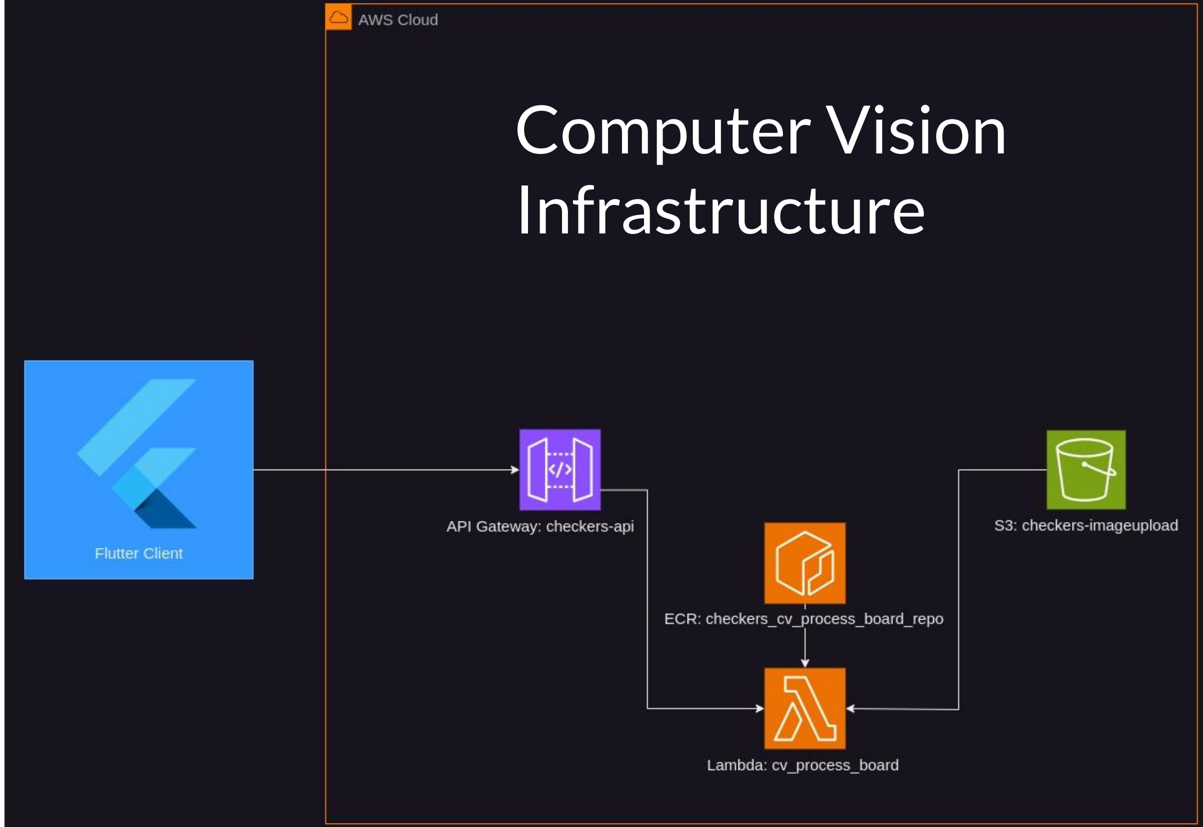
*All Lambdas except one...

Infrastructure

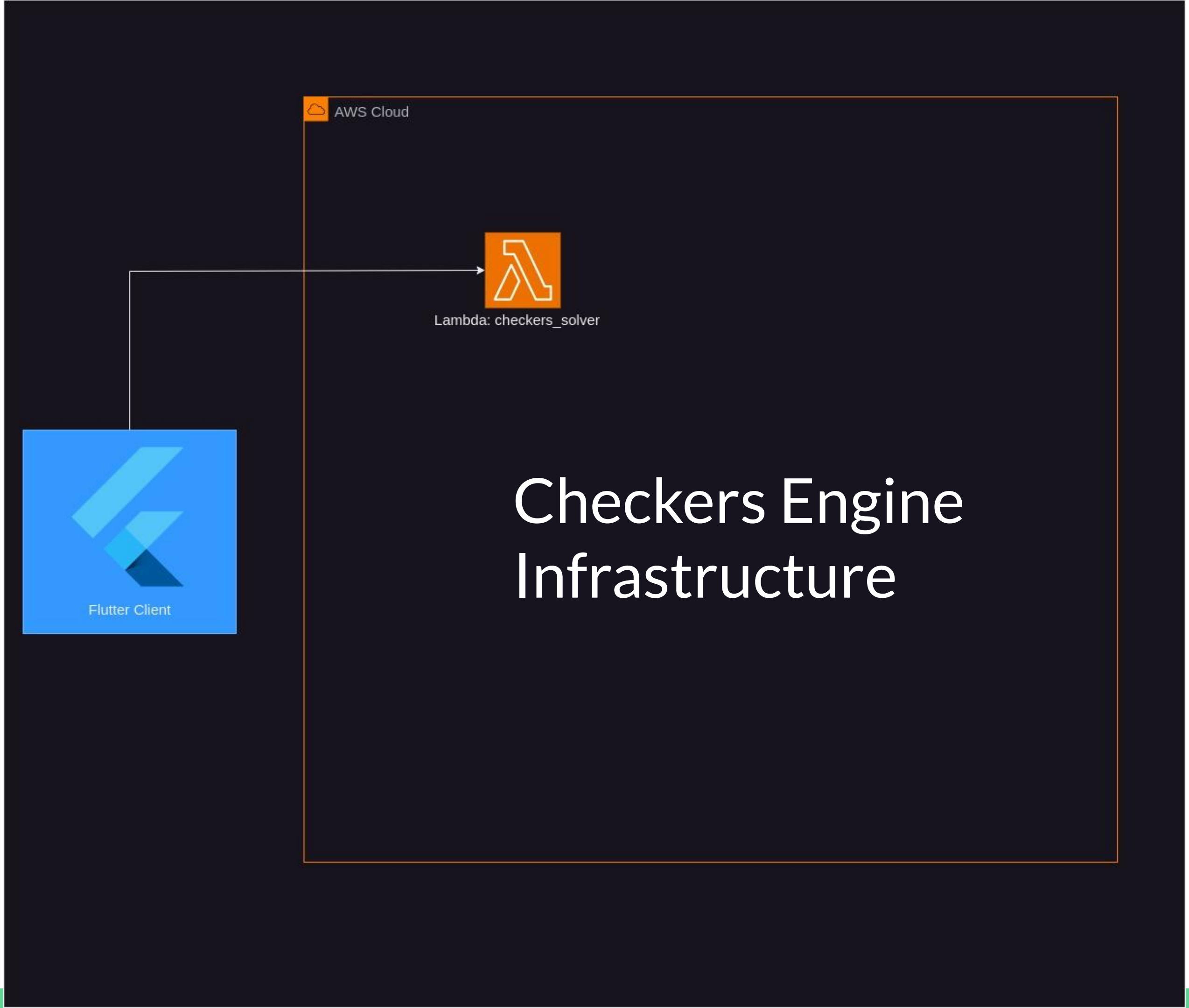



AWS Cloud
ECR: image_upload_repo
Lambda: checkers-image-upload
Flutter Client
API Gateway: checkers-api
S3: checkers-imageupload
Presigned-URL gen.
Infrastructure

Computer Vision Infrastructure



Checkers Engine Infrastructure



Infrastructure - Challenges

- Designing **serverless** architecture
- **Integration** of multiple AWS Services
- **Authentication** implementation
- Working without a **deployment pipeline**
- Lambda deployments in **different runtimes** - Java & Python3
    - “Lambda-ifying”
    - Maven build for Checkers Engine
        - Packaging dependencies

Amazon Elastic Container Registry

JUnit for JAVA testing

Maven build system for JAVA

Accomplishments

Frontend

- Seamless user experience
- **Deployment across 4 platforms**
    - Android, iOS, Chrome (web), Linux

Checkers Engine

- **Ability to get next optimal move (MCTS)**
- > 93% code coverage
- Ability to parse the game state from the frontend

Computer Vision

- **Board detection**
- Piece detection
- Took steps to ensure correctness

Infrastructure

- **Serverless**
- Integration between AWS API Gateway, AWS Lambdas, AWS S3, AWS ECR, and GitLab CI/CD
- **No user registration**

Contributions

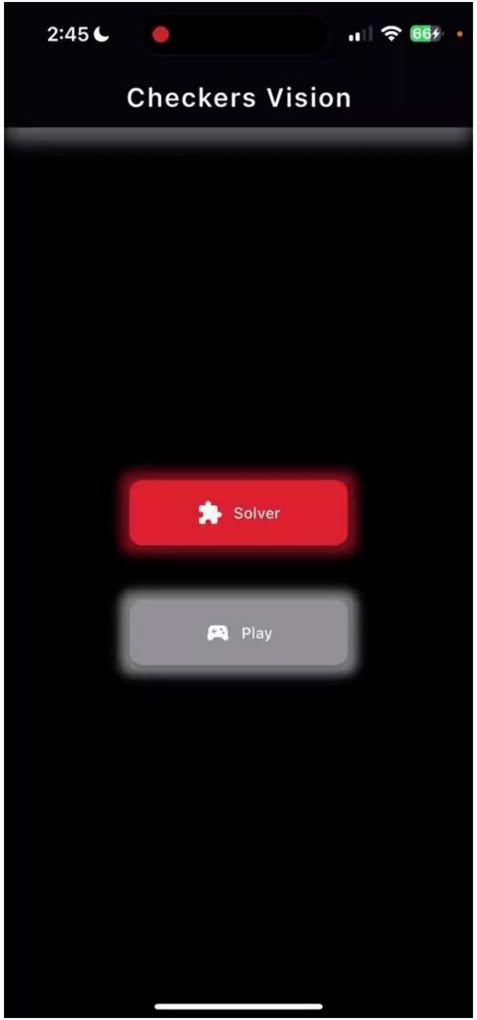
Sibhat Yosef
- Frontend - Solver screen, Board building, Reset button, Play From Here button, Solve button
  - Traversal through pages
- Backend communication with Checkers Engine Lambda

Siddhant Jaju
- Implementation of Checkers Engine for Player and AI
- Logic and Testing the Checkers Engine
- Coverage for all classes in the Checkers Engine
- Methods to parse data from frontend

Yaaseen Basha
- CV - Board detection, Anomaly processing, King/Piece detection
- Frontend - Play screen, Game logic, Hint button, Against AI, Highlighting move(s)

Ranai Srivastav
- CV - Anomaly processing, Piece detection, return data structure
- Infrastructure - “Lambda-ifying” CV, Checkers Engine, maintenance
- Infrastructure - Researching, identifying AWS products to use for application

Charles Dudley
- AWS - Research, design, & integration for infrastructure
- CI/CD - Gitlab Runner deployment pipeline
- Frontend - Triggering CV


2:45
Checkers Vision
Solver
Play

Questions?