

Bachelorthesis

Generated by Doxygen 1.9.1

1 Bachelor-Thesis-Project	1
1.1 Build Instructions	1
1.2 Camera Calibration	1
1.3 Marker Generation	1
1.4 Run Program	1
2 Namespace Index	3
2.1 Namespace List	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
5.1 badthings Namespace Reference	9
5.1.1 Detailed Description	9
5.1.2 Function Documentation	9
5.1.2.1 emptyBadThings()	9
5.1.2.2 looseClass()	10
5.1.2.3 looseHand()	10
5.1.2.4 looseLevel()	10
5.1.2.5 maleDeadFemaleLevelDown()	11
5.1.2.6 playerDies()	11
5.2 cardtypeaction Namespace Reference	11
5.2.1 Detailed Description	12
5.2.2 Function Documentation	12
5.2.2.1 curse()	12
5.2.2.2 item()	12
5.2.2.3 itemBuff()	13
5.2.2.4 joker()	13
5.2.2.5 lvlUp()	13
5.2.2.6 monster()	13
5.2.2.7 munchClass()	14
5.2.2.8 race()	14
5.3 constants Namespace Reference	14
5.3.1 Variable Documentation	15
5.3.1.1 yolo_grid_size	15
5.4 extras Namespace Reference	15
5.5 munch_tut Namespace Reference	15
5.5.1 Detailed Description	15
5.5.2 Typedef Documentation	15
5.5.2.1 Color	15

6 Class Documentation	17
6.1 BadThingsRetVal Struct Reference	17
6.2 Button Class Reference	17
6.2.1 Detailed Description	18
6.2.2 Constructor & Destructor Documentation	18
6.2.2.1 Button() [1/3]	18
6.2.2.2 Button() [2/3]	19
6.2.2.3 Button() [3/3]	19
6.2.3 Member Function Documentation	19
6.2.3.1 draw()	19
6.2.3.2 operator=() [1/2]	19
6.2.3.3 operator=() [2/2]	19
6.2.3.4 poll_click()	20
6.2.4 Member Data Documentation	20
6.2.4.1 callback	20
6.2.4.2 color	20
6.2.4.3 id	20
6.2.4.4 origin	20
6.2.4.5 rect	21
6.2.4.6 visible	21
6.3 CardTypeRetVal Struct Reference	21
6.4 ExtrasRetVal Struct Reference	21
6.5 GameState Struct Reference	21
6.5.1 Detailed Description	22
6.5.2 Member Data Documentation	22
6.5.2.1 buttons	22
6.5.2.2 canvas_size	22
6.5.2.3 end_turn	23
6.5.2.4 mouseparams	23
6.5.2.5 nnResults	23
6.5.2.6 player01	23
6.5.2.7 remove_card	23
6.5.2.8 run_away	23
6.5.2.9 should_continue	24
6.5.2.10 should_exit	24
6.6 InputEvent Struct Reference	24
6.6.1 Member Data Documentation	24
6.6.1.1 type	24
6.7 MouseParams Struct Reference	24
6.7.1 Detailed Description	25
6.7.2 Member Data Documentation	25
6.7.2.1 clickP	25

6.7.2.2 event	25
6.7.2.3 markerCorner	25
6.7.2.4 markerId	26
6.7.2.5 poly	26
6.7.2.6 tutText	26
6.8 MunchkinCard Class Reference	26
6.8.1 Detailed Description	27
6.8.2 Constructor & Destructor Documentation	28
6.8.2.1 MunchkinCard() [1/4]	28
6.8.2.2 MunchkinCard() [2/4]	28
6.8.2.3 MunchkinCard() [3/4]	29
6.8.2.4 MunchkinCard() [4/4]	29
6.8.3 Member Function Documentation	29
6.8.3.1 cardsConstr()	29
6.8.3.2 operator=() [1/2]	29
6.8.3.3 operator=() [2/2]	30
6.8.4 Member Data Documentation	30
6.8.4.1 badThings	30
6.8.4.2 badThingsFunc	30
6.8.4.3 bonis	30
6.8.4.4 cardName	30
6.8.4.5 debuff	30
6.8.4.6 effect	31
6.8.4.7 handsNeeded	31
6.8.4.8 itemEffect	31
6.8.4.9 itemLarge	31
6.8.4.10 itemNeeds	31
6.8.4.11 itemType	31
6.8.4.12 itemValue	32
6.8.4.13 lvUp	32
6.8.4.14 markerID	32
6.8.4.15 monsStrength	32
6.8.4.16 parentCardType	32
6.8.4.17 strengthBoni	32
6.8.4.18 treasures	33
6.8.4.19 type	33
6.9 NNresult Struct Reference	33
6.9.1 Detailed Description	33
6.9.2 Member Data Documentation	33
6.9.2.1 id	33
6.9.2.2 nnBBox	34
6.10 PlayerStats Struct Reference	34

6.10.1 Detailed Description	35
6.10.2 Member Data Documentation	35
6.10.2.1 availableClasses	35
6.10.2.2 availableRaces	35
6.10.2.3 bonis	35
6.10.2.4 carriesLargeItems	35
6.10.2.5 curses	35
6.10.2.6 hands	36
6.10.2.7 hasArmor	36
6.10.2.8 hasHat	36
6.10.2.9 hasShoes	36
6.10.2.10 itemEffects	36
6.10.2.11 lvl	36
6.10.2.12 munchClasses	37
6.10.2.13 munchRaces	37
6.10.2.14 runStrength	37
6.10.2.15 sex	37
6.10.2.16 strength	37
6.11 testOutput Struct Reference	37
6.11.1 Detailed Description	38
6.11.2 Member Data Documentation	38
6.11.2.1 allClasses	38
6.11.2.2 allPropability	38
6.11.2.3 csvFileString	39
6.11.2.4 csvSingleFile	39
6.11.2.5 debugSingleOutput	39
6.11.2.6 debugTick	39
6.11.2.7 fileCounter	39
6.11.2.8 fileString	39
6.11.2.9 inferenceTime	40
6.11.2.10 MunchkinClassesNum	40
6.11.2.11 singleFile	40
6.11.2.12 YoloName	40
7 File Documentation	41
7.1 ArucoGenerator.cpp File Reference	41
7.1.1 Function Documentation	41
7.1.1.1 createArucoMarkers()	41
7.2 ArucoGenerator.h File Reference	41
7.2.1 Detailed Description	42
7.2.2 Function Documentation	42
7.2.2.1 createArucoMarkers()	42

7.3 BadThings.cpp File Reference	42
7.4 BadThings.h File Reference	42
7.4.1 Detailed Description	43
7.4.2 Typedef Documentation	43
7.4.2.1 BadThingsFunc	43
7.5 Button.cpp File Reference	44
7.6 Button.h File Reference	44
7.6.1 Detailed Description	44
7.6.2 Typedef Documentation	44
7.6.2.1 button_callback	45
7.6.3 Enumeration Type Documentation	45
7.6.3.1 ButtonOrigin	45
7.7 CameraCalibration.cpp File Reference	45
7.7.1 Function Documentation	46
7.7.1.1 cameraCalibration()	46
7.7.1.2 cameraCalibrationProcess()	46
7.7.1.3 createKnownBoardPositions()	46
7.7.1.4 getChessboardCorners()	47
7.7.1.5 loadCameraCalibration()	47
7.7.1.6 saveCameraCalibration()	48
7.8 CameraCalibration.h File Reference	48
7.8.1 Detailed Description	49
7.8.2 Function Documentation	49
7.8.2.1 cameraCalibration()	49
7.8.2.2 cameraCalibrationProcess()	49
7.8.2.3 createKnownBoardPositions()	50
7.8.2.4 getChessboardCorners()	50
7.8.2.5 loadCameraCalibration()	50
7.8.2.6 saveCameraCalibration()	51
7.9 CardTypeActions.cpp File Reference	51
7.9.1 Function Documentation	52
7.9.1.1 runAway()	52
7.10 CardTypeActions.h File Reference	52
7.10.1 Detailed Description	53
7.10.2 Typedef Documentation	53
7.10.2.1 CardTypeFunc	53
7.11 common.h File Reference	53
7.11.1 Detailed Description	54
7.12 DetectionVariants.cpp File Reference	54
7.12.1 Function Documentation	54
7.12.1.1 ArucoMatching()	54
7.12.1.2 YOLOMatching()	55

7.13 DetectionVariants.h File Reference	55
7.13.1 Detailed Description	56
7.13.2 Function Documentation	56
7.13.2.1 ArucoMatching()	56
7.13.2.2 YOLOMatching()	57
7.14 Extras.cpp File Reference	57
7.15 Extras.h File Reference	57
7.15.1 Typedef Documentation	58
7.15.1.1 ExtrasFunc	58
7.16 GameState.h File Reference	58
7.16.1 Detailed Description	59
7.16.2 Enumeration Type Documentation	59
7.16.2.1 MouseEvent	59
7.17 InputEvent.h File Reference	59
7.17.1 Detailed Description	60
7.17.2 Enumeration Type Documentation	60
7.17.2.1 EventType	60
7.18 main.cpp File Reference	60
7.18.1 Function Documentation	61
7.18.1.1 button_continue()	62
7.18.1.2 button_end_turn()	62
7.18.1.3 button_exit()	62
7.18.1.4 callBackFunction()	62
7.18.1.5 logic()	62
7.18.1.6 main()	62
7.18.1.7 startWebcamMonitoring()	63
7.18.1.8 testArucoMatching()	63
7.18.1.9 TestYOLOMatching()	63
7.18.2 Variable Documentation	63
7.18.2.1 arucoSquareDimension	63
7.18.2.2 calbirationSquareDimension	63
7.18.2.3 cards	63
7.18.2.4 chessboardDimensions	64
7.18.2.5 classes	64
7.18.2.6 confThreshold	64
7.18.2.7 debugSingleOutput	64
7.18.2.8 functMap	64
7.18.2.9 gamestate	64
7.18.2.10 modus	64
7.18.2.11 multipleTests	64
7.18.2.12 MunchkinClassesNum	65
7.18.2.13 nmsThreshold	65

7.18.2.14 testNum	65
7.18.2.15 YoloName	65
7.19 MunchkinCards.cpp File Reference	65
7.20 MunchkinCards.h File Reference	65
7.20.1 Detailed Description	66
7.20.2 Enumeration Type Documentation	66
7.20.2.1 CardType	66
7.20.2.2 ItemType	67
7.20.2.3 ParentCardType	67
7.21 PostProcessFunctions.cpp File Reference	67
7.21.1 Function Documentation	68
7.21.1.1 chooseCardColor()	68
7.21.1.2 drawPred()	68
7.21.1.3 getOutputsNames()	69
7.21.1.4 postprocess()	69
7.22 PostProcessFunctions.h File Reference	70
7.22.1 Detailed Description	71
7.22.2 Function Documentation	71
7.22.2.1 chooseCardColor()	71
7.22.2.2 drawPred()	71
7.22.2.3 getOutputsNames()	72
7.22.2.4 postprocess()	72
7.23 README.md File Reference	73
7.24 TTMunchkinTut.cpp File Reference	73
7.25 YOLODataGenerator.cpp File Reference	73
7.25.1 Function Documentation	73
7.25.1.1 generateDatasetNN()	74
7.25.1.2 uninorm()	74
7.25.1.3 unirand()	74
7.26 YOLODataGenerator.h File Reference	74
7.26.1 Detailed Description	75
7.26.2 Function Documentation	75
7.26.2.1 generateDatasetNN()	75
7.26.2.2 uninorm()	75
7.26.2.3 unirand()	75

Chapter 1

Bachelor-Thesis-Project

1.1 Build Instructions

OpenCV 3 with contrib modules needs to be installed via CMake. Add a environment variable "OpenCV_DIR" to the path of the corresponding install root directory. Then build the project itself as usual.

1.2 Camera Calibration

To calibrate Camera use pattern.png on a piece of cardboard. To start Calibration add "-c" Commandline Option. To Calibrate you need at least 15 valid pictures of Chessboard pattern. To save a picture while in calibration mode press Space. After at least 15 valid Pictures (around 50 are recommended) press Enter to start calibration Process.

1.3 Marker Generation

To generate Aruco Markers add "-g" Commandline Option. You can Change quantity of Markers and Markerlibrary in the createArucoMarker() Function (file `main.cpp`).

1.4 Run Program

After you generated Aruco Markers and printed them out you can remove Commandline Options and start the Program. Aruco Markers 1-15 (except 4-6) are implemented atm. Corresponding Munchkincards can be found in `MunchkinCards.cpp`.

There are some bugs left since this is only a Proof of Concept. Sometimes `Button` have to be pressed multiple Times to work. Random Generator to run away mostly lets you run away.

<https://th-koeln.sciebo.de/s/9oSsSW6dSUsiHBE>

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

badthings	Namespace containing all bad things functions. These can be assigned to a specific card . . .	9
cardtypeaction	Namespace containing all cardtype functions. These are mapped to the actual card type . . .	11
constants	14
extras	15
munch_tut	Namespace for munchkin tutorial	15

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

BadThingsRetVal	17
Button	
Class for Buttons	17
CardTypeRetVal	21
ExtrasRetVal	21
GameState	21
InputEvent	24
MouseParams	
Holds data of a mouse event	24
MunchkinCard	
Class of munchkin cards with params that could be needed for a single card	26
NNresult	
Struct that holds custom result from YOLO Neuronal Net	33
PlayerStats	
Holds data of player stats	34
testOutput	
Struct for object that is used for the YOLO tests	37

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

ArucoGenerator.cpp	41
ArucoGenerator.h	
Class to generate new Aruco Markers	41
BadThings.cpp	42
BadThings.h	
Bad Things Functions for Munchkin Cards	42
Button.cpp	44
Button.h	
Class to define buttons which are used for user input	44
CameraCalibration.cpp	45
CameraCalibration.h	
Camera Calibration with OpenCV Chessboard	48
CardTypeActions.cpp	51
CardTypeActions.h	
Class to define the functions for munchkin card types	52
common.h	
Class for common used tools, objects, etc	53
DetectionVariants.cpp	54
DetectionVariants.h	
Multiple Detection Variants for Munchkin Tutorial	55
Extras.cpp	57
Extras.h	57
GameState.h	
GameState class to store all needed parameters for tutorial	58
InputEvent.h	
Class for all input events of the player through mouseclicks	59
main.cpp	60
MunchkinCards.cpp	65
MunchkinCards.h	
MunchkinCards class in which all parameters for the munchkin cards are stored	65
PostProcessFunctions.cpp	67
PostProcessFunctions.h	
All util/post process functions	70
TTMunchkinTut.cpp	73
YOLODataGenerator.cpp	73
YOLODataGenerator.h	
Generates Data for YOLO training	74

Chapter 5

Namespace Documentation

5.1 badthings Namespace Reference

Namespace containing all bad things functions. These can be assigned to a specific card.

Functions

- [BadThingsRetVal emptyBadThings](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &[card](#))
Does nothing (placeholder)
- [BadThingsRetVal looseHand](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &[card](#))
Player loses 1 hand and the function checks whether or not he can still carry all his items that need hands.
- [BadThingsRetVal looseClass](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &[card](#))
Player loses his class.
- [BadThingsRetVal playerDies](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &[card](#))
Player dies.
- [BadThingsRetVal looseLevel](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &[card](#))
Player loses 1 level.
- [BadThingsRetVal maleDeadFemaleLevelDown](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &[card](#))
Male Players die. Female players loose 1 level.

5.1.1 Detailed Description

Namespace containing all bad things functions. These can be assigned to a specific card.

5.1.2 Function Documentation

5.1.2.1 emptyBadThings()

```
BadThingsRetVal badthings::emptyBadThings (  
    GameState & gamestate,  
    const MunchkinCard & card )
```

Does nothing (placeholder)

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.1.2.2 looseClass()

```
BadThingsRetVal badthings::looseClass (
    GameState & gamestate,
    const MunchkinCard & card )
```

Player loses his class.

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.1.2.3 looseHand()

```
BadThingsRetVal badthings::looseHand (
    GameState & gamestate,
    const MunchkinCard & card )
```

Player loses 1 hand and the function checks whether or not he can still carry all his items that need hands.

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.1.2.4 looseLevel()

```
BadThingsRetVal badthings::looseLevel (
    GameState & gamestate,
    const MunchkinCard & card )
```

Player loses 1 level.

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.1.2.5 maleDeadFemaleLevelDown()

```
BadThingsRetVal badthings::maleDeadFemaleLevelDown (
    GameState & gamestate,
    const MunchkinCard & card )
```

Male Players die. Female players loose 1 level.

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.1.2.6 playerDies()

```
BadThingsRetVal badthings::playerDies (
    GameState & gamestate,
    const MunchkinCard & card )
```

Player dies.

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.2 cardtypeaction Namespace Reference

Namespace containing all cardtype functions. These are mapped to the actual card type.

Functions

- `CardTypeRetVal curse (GameState &gamestate, const MunchkinCard &card)`
processes card type curse
- `CardTypeRetVal joker (GameState &gamestate, const MunchkinCard &card)`
processes card type joker
- `CardTypeRetVal monster (GameState &gamestate, const MunchkinCard &card)`
proceses card type monster
- `CardTypeRetVal munchClass (GameState &gamestate, const MunchkinCard &card)`
processes card Type munchkin class
- `CardTypeRetVal race (GameState &gamestate, const MunchkinCard &card)`

- processes card type munchkin race*
- `CardTypeRetVal item (GameState &gamestate, const MunchkinCard &card)`
processes card type item
- `CardTypeRetVal itemBuff (GameState &gamestate, const MunchkinCard &card)`
processes card type item buff
- `CardTypeRetVal lvlUp (GameState &gamestate, const MunchkinCard &card)`
processes card type level up

5.2.1 Detailed Description

Namespace containing all cardtype functions. These are mapped to the actual card type.

5.2.2 Function Documentation

5.2.2.1 `curse()`

```
CardTypeRetVal cardtypeaction::curse (
    GameState & gamestate,
    const MunchkinCard & card )
```

processes card type curse

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.2.2.2 `item()`

```
CardTypeRetVal cardtypeaction::item (
    GameState & gamestate,
    const MunchkinCard & card )
```

processes card type item

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.2.2.3 itemBuff()

```
CardTypeRetVal cardtypeaction::itemBuff (
    GameState & gamestate,
    const MunchkinCard & card )
```

processes card type item buff

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.2.2.4 joker()

```
CardTypeRetVal cardtypeaction::joker (
    GameState & gamestate,
    const MunchkinCard & card )
```

processes card type joker

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.2.2.5 lvlUp()

```
CardTypeRetVal cardtypeaction::lvlUp (
    GameState & gamestate,
    const MunchkinCard & card )
```

processes card type level up

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.2.2.6 monster()

```
CardTypeRetVal cardtypeaction::monster (
```

```

    GameState & gamestate,
    const MunchkinCard & card )

```

processes card type monster

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.2.2.7 munchClass()

```

CardTypeRetVal cardtypeaction::munchClass (
    GameState & gamestate,
    const MunchkinCard & card )

```

processes card Type munchkin class

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.2.2.8 race()

```

CardTypeRetVal cardtypeaction::race (
    GameState & gamestate,
    const MunchkinCard & card )

```

processes card type munchkin race

Parameters

<i>gamestate</i>	The game state is modified according to the corresponding munchkin rule/card.
<i>card</i>	Used to supply additional information about the card.

5.3 constants Namespace Reference

Variables

- constexpr int `yolo_grid_size` = 13

5.3.1 Variable Documentation

5.3.1.1 yolo_grid_size

```
constexpr int constants::yolo_grid_size = 13  [inline], [constexpr]
```

5.4 extras Namespace Reference

5.5 munch_tut Namespace Reference

namespace for munchkin tutorial

Typedefs

- using [Color](#) = std::array< int, 3 >

5.5.1 Detailed Description

namespace for munchkin tutorial

5.5.2 Typedef Documentation

5.5.2.1 Color

```
using munch_tut::Color = typedef std::array<int, 3>
```


Chapter 6

Class Documentation

6.1 BadThingsRetVal Struct Reference

```
#include <BadThings.h>
```

The documentation for this struct was generated from the following file:

- [BadThings.h](#)

6.2 Button Class Reference

class for Buttons

```
#include <Button.h>
```

Public Member Functions

- [Button](#) (int _id, const cv::Rect &_rect, const cv::Scalar &_color, bool _visible=true, [ButtonOrigin](#) _↔
origin=[ButtonOrigin::topleft](#), const [button_callback](#) &_callback={})
constructor for a button
- [Button](#) (const [Button](#) &)=default
Default copy constructor.
- [Button](#) ([Button](#) &&)=default
Default move constructor.
- [Button](#) & [operator=](#) (const [Button](#) &)=default
Default copy assignment.
- [Button](#) & [operator=](#) ([Button](#) &&)=default
Default move assignment.
- bool [poll_click](#) (const cv::Point &point, const cv::Size &canvas_size) const
function to check if click was inside the button
- void [draw](#) (cv::Mat &canvas) const
draw function for buttons

Public Attributes

- int [id](#)
Button id.
- cv::Rect [rect](#)
rectangle that defines size of button
- [button_callback](#) [callback](#)
callback function for button to apply action to button
- cv::Scalar [color](#)
color of the button
- bool [visible](#)
wether or not button should be visible
- [ButtonOrigin](#) [origin](#)
position of the button

6.2.1 Detailed Description

class for Buttons

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Button() [1/3]

```
Button::Button (
    int _id,
    const cv::Rect & _rect,
    const cv::Scalar & _color,
    bool _visible = true,
    ButtonOrigin _origin = ButtonOrigin::topleft,
    const button\_callback & _callback = {} )
```

constructor for a button

Parameters

_id	Button id
_rect	rectangle which defines the dimensions of the button
_color	defines the color of the button in GBR
_visible	used to hide button when not needed
_origin	origin of the button as in enum class ButtonOrigin
_callback	callback function for button. Can be changed so that one button can serve mulitple purposes if necessary

6.2.2.2 Button() [2/3]

```
Button::Button (
    const Button & ) [default]
```

Default copy constructor.

6.2.2.3 Button() [3/3]

```
Button::Button (
    Button && ) [default]
```

Default move constructor.

6.2.3 Member Function Documentation

6.2.3.1 draw()

```
void Button::draw (
    cv::Mat & canvas ) const
```

draw function for buttons

Parameters

<i>canvas</i>	defines the button that should be drawn
---------------	---

6.2.3.2 operator=() [1/2]

```
Button& Button::operator= (
    Button && ) [default]
```

Default move assignment.

6.2.3.3 operator=() [2/2]

```
Button& Button::operator= (
    const Button & ) [default]
```

Default copy assignment.

6.2.3.4 poll_click()

```
bool Button::poll_click (
    const cv::Point & point,
    const cv::Size & canvas_size ) const
```

function to check if click was inside the button

Parameters

<i>point</i>	point of click
<i>canvas_size</i>	size of the canvas that defines the button size

6.2.4 Member Data Documentation

6.2.4.1 callback

[button_callback](#) Button::callback

callback function for button to apply action to button

6.2.4.2 color

`cv::Scalar` Button::color

color of the button

6.2.4.3 id

`int` Button::id

[Button](#) id.

6.2.4.4 origin

[ButtonOrigin](#) Button::origin

position of the button

6.2.4.5 rect

```
cv::Rect Button::rect
```

rectangle that defines size of button

6.2.4.6 visible

```
bool Button::visible
```

wether or not button should be visible

The documentation for this class was generated from the following files:

- [Button.h](#)
- [Button.cpp](#)

6.3 CardTypeRetVal Struct Reference

```
#include <CardTypeActions.h>
```

The documentation for this struct was generated from the following file:

- [CardTypeActions.h](#)

6.4 ExtrasRetVal Struct Reference

```
#include <Extras.h>
```

The documentation for this struct was generated from the following file:

- [Extras.h](#)

6.5 GameState Struct Reference

```
#include <GameState.h>
```

Public Attributes

- [PlayerStats](#) `player01`
playerStats object for data of player stats that are needed
- [MouseParams](#) `mouseparams`
mouseParams object for data of tutorial that are needed e.g. tutorial text
- `std::vector< NNresult > nnResults`
vector of [NNresult](#) objects to store all card found by YOLO Neuronal Net
- `std::vector< Button > buttons`
buttons that are needed for the tutorial
- `bool should_exit`
param wether or not the game should exit after exit button was pressed
- `bool should_continue`
param for when user input is awaited and the tutorial needs to jump to a different point in logic function
- `bool end_turn`
param to signal the player that the end of the turn is reached
- `bool run_away`
param for when the user needs to run away and the tutorial triggers an random number as dice roll
- `bool remove_card`
param for when the player chooses or needs to get rid of a card he has equiped
- `cv::Size canvas_size`
param of the canvas size used for the buttons

6.5.1 Detailed Description

data of all needed params for the tutorial

6.5.2 Member Data Documentation

6.5.2.1 buttons

```
std::vector<Button> GameState::buttons
```

buttons that are needed for the tutorial

6.5.2.2 canvas_size

```
cv::Size GameState::canvas_size
```

param of the canvas size used for the buttons

6.5.2.3 end_turn

```
bool GameState::end_turn
```

param to signal the player that the end of the turn is reached

6.5.2.4 mouseparams

```
MouseParams GameState::mouseparams
```

mouseParams object for data of tutorial that are needed e.g. tutorial text

6.5.2.5 nnResults

```
std::vector<NNresult> GameState::nnResults
```

vector of [NNresult](#) objects to store all card found by YOLO Neuronal Net

6.5.2.6 player01

```
PlayerStats GameState::player01
```

playerStats object for data of player stats that are needed

6.5.2.7 remove_card

```
bool GameState::remove_card
```

param for when the player chooses or needs to get rid of a card he has equipped

6.5.2.8 run_away

```
bool GameState::run_away
```

param for when the user needs to run away and the tutorial triggers an random number as dice roll

6.5.2.9 should_continue

```
bool GameState::should_continue
```

param for when user input is awaited and the tutorial needs to jump to a different point in logic function

6.5.2.10 should_exit

```
bool GameState::should_exit
```

param wether or not the game should exit after exit button was pressed

The documentation for this struct was generated from the following file:

- [GameState.h](#)

6.6 InputEvent Struct Reference

```
#include <InputEvent.h>
```

Public Attributes

- [EventType](#) type

6.6.1 Member Data Documentation

6.6.1.1 type

```
EventType InputEvent::type
```

The documentation for this struct was generated from the following file:

- [InputEvent.h](#)

6.7 MouseParams Struct Reference

Holds data of a mouse event.

```
#include <GameState.h>
```

Public Attributes

- `vector< cv::Point > poly`
Polygon of the clicked marker.
- `int markerId`
Id of the marker.
- `vector< cv::Point2f > markerCorner`
Four corners of the marker.
- `vector< string > tutText`
Tutorial text to display for the marker.
- `cv::Point clickP`
Mouse position.
- `MouseEvent event`
Mouse event type.

6.7.1 Detailed Description

Holds data of a mouse event.

6.7.2 Member Data Documentation

6.7.2.1 clickP

```
cv::Point MouseParams::clickP
```

Mouse position.

6.7.2.2 event

```
MouseEvent MouseParams::event
```

Mouse event type.

6.7.2.3 markerCorner

```
vector<cv::Point2f> MouseParams::markerCorner
```

Four corners of the marker.

6.7.2.4 markerId

```
int MouseParams::markerId
```

Id of the marker.

6.7.2.5 poly

```
vector<cv::Point> MouseParams::poly
```

Polygon of the clicked marker.

6.7.2.6 tutText

```
vector<string> MouseParams::tutText
```

Tutorial text to display for the marker.

The documentation for this struct was generated from the following file:

- [GameState.h](#)

6.8 MunchkinCard Class Reference

class of munchkin cards with params that could be needed for a single card

```
#include <MunchkinCards.h>
```

Public Member Functions

- [MunchkinCard](#) ()
Default constructor.
- [MunchkinCard](#) (int _markerID, const string &_cardName, const string &_effect, const string &_badThings, const string &_itemEffect, const string &_itemNeeds, vector< string > _bonis, [ParentCardType](#) _parent←CardType, [CardType](#) _type, [ItemType](#) _itemType, int _strengthBoni, int _debuff, int _monsStrength, int _lvlUp, int _treasures, int _itemValue, int _handsNeeded, bool _itemLarge)
custom constructor
- [MunchkinCard](#) (const [MunchkinCard](#) &other)=default
default copy constructor
- [MunchkinCard](#) ([MunchkinCard](#) &&other)=default
default move constructor
- [MunchkinCard](#) & operator= (const [MunchkinCard](#) &other)=default
default copy assignment
- [MunchkinCard](#) & operator= ([MunchkinCard](#) &&other)=default
default move assignment

Static Public Member Functions

- static vector< [MunchkinCard](#) > [cardsConstr](#) ()
function to create all cards from [MunchkinCards.cpp](#)

Public Attributes

- string [cardName](#)
card name
- string [effect](#)
card effect
- vector< string > [bonis](#)
boni that the card can have
- string [badThings](#)
(deprecated) string of bad things
- string [itemEffect](#)
effect that an item has e.g. attack with fire and flame
- string [itemNeeds](#)
param that a munchkin needs to equip a item e.g. has to be a dwarf
- [ParentCardType](#) [parentCardType](#)
param for parent card type
- [CardType](#) [type](#)
param for "child" card type
- [ItemType](#) [itemType](#)
item type
- int [markerID](#)
marker id that corresponse with the card
- int [strengthBoni](#)
how much strength the munchkin receives with this card
- int [debuff](#)
how much strength the munchkin loses through this card
- int [monsStrength](#)
param how much the monsters strength is
- int [lvlUp](#)
how many level you get r.g. when defeating the monster
- int [treasures](#)
how many treasure the munchkin gets when defeating the monster
- int [itemValue](#)
how much gold the item sells for
- int [handsNeeded](#)
how many hands a munchkin needs to equip the item
- bool [itemLarge](#)
wether or not the item is large
- [BadThingsFunc](#) [badThingsFunc](#)
bad things function that mapps onto a bad things function in [badThingsFunc.h](#)

6.8.1 Detailed Description

class of munchkin cards with params that could be needed for a single card

6.8.2 Constructor & Destructor Documentation

6.8.2.1 MunchkinCard() [1/4]

```
MunchkinCard::MunchkinCard ( ) [inline]
```

Default constructor.

6.8.2.2 MunchkinCard() [2/4]

```
MunchkinCard::MunchkinCard (
    int _markerID,
    const string & _cardName,
    const string & _effect,
    const string & _badThings,
    const string & _itemEffect,
    const string & _itemNeeds,
    vector< string > _bonis,
    ParentCardType _parentCardType,
    CardType _type,
    ItemType _itemType,
    int _strengthBoni,
    int _debuff,
    int _monsStrength,
    int _lvlUp,
    int _treasures,
    int _itemValue,
    int _handsNeeded,
    bool _itemLarge )
```

custom constructor

Parameters

<i>_markerID</i>	marker id that corresponsse with the card
<i>_cardName</i>	card name
<i>_effect</i>	card effect
<i>_badThings</i>	(deprecated) string of bad things
<i>_itemEffect</i>	effect that an item has e.g. attack with fire and flame
<i>_itemNeeds</i>	param that a munchkin needs to equip a item e.g. has to be a dwarf
<i>_bonis</i>	boni that the card can have
<i>_parentCardType</i>	param for parent card type
<i>_type</i>	param for "child" card type
<i>_itemType</i>	item type
<i>_strengthBoni</i>	how much strength the munchkin receives with this card
<i>_debuff</i>	how much strength the munchkin looses through this card
<i>_monsStrength</i>	param how much the monsters strength is

Parameters

<code>_lvlUp</code>	how many level you get r.g. when defeating the monster
<code>_treasures</code>	how many treasure the munchkin gets when defeating the monster
<code>_itemValue</code>	how much gold the item sells for
<code>_handsNeeded</code>	how many hands a munchkin needs to equip the item
<code>_itemLarge</code>	wether or not the item is large

6.8.2.3 MunchkinCard() [3/4]

```
MunchkinCard::MunchkinCard (
    const MunchkinCard & other ) [default]
```

default copy constructor

6.8.2.4 MunchkinCard() [4/4]

```
MunchkinCard::MunchkinCard (
    MunchkinCard && other ) [default]
```

default move constructor

6.8.3 Member Function Documentation**6.8.3.1 cardsConstr()**

```
vector< MunchkinCard > MunchkinCard::cardsConstr ( ) [static]
```

function to create all cards from [MunchkinCards.cpp](#)

6.8.3.2 operator=() [1/2]

```
MunchkinCard& MunchkinCard::operator= (
    const MunchkinCard & other ) [default]
```

default copy assignment

6.8.3.3 operator=() [2/2]

```
MunchkinCard& MunchkinCard::operator= (
    MunchkinCard && other ) [default]
```

default move assignment

6.8.4 Member Data Documentation

6.8.4.1 badThings

```
string MunchkinCard::badThings
```

(deprecated) string of bad things

6.8.4.2 badThingsFunc

```
BadThingsFunc MunchkinCard::badThingsFunc
```

bad things function that maps onto a bad things function in badThingsFunc.h

6.8.4.3 bonis

```
vector<string> MunchkinCard::bonis
```

boni that the card can have

6.8.4.4 cardName

```
string MunchkinCard::cardName
```

card name

6.8.4.5 debuff

```
int MunchkinCard::debuff
```

how much strength the munchkin loses through this card

6.8.4.6 effect

```
string MunchkinCard::effect
```

card effect

6.8.4.7 handsNeeded

```
int MunchkinCard::handsNeeded
```

how many hands a munchkin needs to equip the item

6.8.4.8 itemEffect

```
string MunchkinCard::itemEffect
```

effect that an item has e.g. attack with fire and flame

6.8.4.9 itemLarge

```
bool MunchkinCard::itemLarge
```

wether or not the item is large

6.8.4.10 itemNeeds

```
string MunchkinCard::itemNeeds
```

param that a munchkin needs to equip a item e.g. has to be a dwarf

6.8.4.11 itemType

```
ItemType MunchkinCard::itemType
```

item type

6.8.4.12 itemValue

```
int MunchkinCard::itemValue
```

how much gold the item sells for

6.8.4.13 lvlUp

```
int MunchkinCard::lvlUp
```

how many level you get r.g. when defeating the monster

6.8.4.14 markerID

```
int MunchkinCard::markerID
```

marker id that correspond with the card

6.8.4.15 monsStrength

```
int MunchkinCard::monsStrength
```

param how much the monsters strength is

6.8.4.16 parentCardType

```
ParentCardType MunchkinCard::parentCardType
```

param for parent card type

6.8.4.17 strengthBoni

```
int MunchkinCard::strengthBoni
```

how much strength the munchkin receives with this card

6.8.4.18 treasures

```
int MunchkinCard::treasures
```

how many treasure the munchkin gets when defeating the monster

6.8.4.19 type

```
CardType MunchkinCard::type
```

param for "child" card type

The documentation for this class was generated from the following files:

- [MunchkinCards.h](#)
- [MunchkinCards.cpp](#)

6.9 NNresult Struct Reference

struct that holds custom result from YOLO Neuronal Net

```
#include <GameState.h>
```

Public Attributes

- `int id`
id of NN result class which translates to recognized card
- `cv::Rect nnBBox`
bounding box of recognized NN result which highlights the card

6.9.1 Detailed Description

struct that holds custom result from YOLO Neuronal Net

6.9.2 Member Data Documentation

6.9.2.1 id

```
int NNresult::id
```

id of NN result class which translates to recognized card

6.9.2.2 nnBBox

```
cv::Rect NNresult::nnBBox
```

bounding box of recognized NN result which highlights the card

The documentation for this struct was generated from the following file:

- [GameState.h](#)

6.10 PlayerStats Struct Reference

Holds data of player stats.

```
#include <GameState.h>
```

Public Attributes

- string [sex](#)
sex of player (male/female)
- vector< string > [bonis](#)
bonis for player that are gained through different cards e.g. munchkin class
- vector< string > [curses](#)
permanent curses that are a burden for the player
- vector< string > [itemEffects](#)
effects of different items that are supporting the player e.g. attack with fire
- vector< string > [munchClasses](#)
munchkin classes the player obtained
- vector< string > [munchRaces](#)
munchkin races the player obtained
- int [lvl](#)
level of the player
- int [strength](#)
strength of a player through his equipment
- int [hands](#)
how many hands player has left that are not holding an object
- int [runStrength](#)
dice roll hat to be larger than x for the player to run away
- int [availableClasses](#)
how many classes can the player obtain
- int [availableRaces](#)
how many races can the player obtain
- bool [carriesLargeItems](#)
wether or not the player carries a large item
- bool [hasArmor](#)
wether or not the player carries an armor
- bool [hasHat](#)
wether or not the player carries a hat
- bool [hasShoes](#)
wether or not the player carries shoes

6.10.1 Detailed Description

Holds data of player stats.

6.10.2 Member Data Documentation

6.10.2.1 availableClasses

```
int PlayerStats::availableClasses
```

how many classes can the player obtain

6.10.2.2 availableRaces

```
int PlayerStats::availableRaces
```

how many races can the player obtain

6.10.2.3 bonis

```
vector<string> PlayerStats::bonis
```

bonis for player that are gained through different cards e.g. munchkin class

6.10.2.4 carriesLargeItems

```
bool PlayerStats::carriesLargeItems
```

wether or not the player carries a large item

6.10.2.5 curses

```
vector<string> PlayerStats::curses
```

permanent curses that are a burden for the player

6.10.2.6 hands

```
int PlayerStats::hands
```

how many hands player has left that are not holding an object

6.10.2.7 hasArmor

```
bool PlayerStats::hasArmor
```

wether or not the player carries an armor

6.10.2.8 hasHat

```
bool PlayerStats::hasHat
```

wether or not the player carries a hat

6.10.2.9 hasShoes

```
bool PlayerStats::hasShoes
```

wether or not the player carries shoes

6.10.2.10 itemEffects

```
vector<string> PlayerStats::itemEffects
```

effects of different items that are supporting the player e.g. attack with fire

6.10.2.11 lvl

```
int PlayerStats::lvl
```

level of the player

6.10.2.12 munchClasses

```
vector<string> PlayerStats::munchClasses
```

munchkin classes the player obtained

6.10.2.13 munchRaces

```
vector<string> PlayerStats::munchRaces
```

munchkin races the player obtained

6.10.2.14 runStrength

```
int PlayerStats::runStrength
```

dice roll has to be larger than x for the player to run away

6.10.2.15 sex

```
string PlayerStats::sex
```

sex of player (male/female)

6.10.2.16 strength

```
int PlayerStats::strength
```

strength of a player through his equipment

The documentation for this struct was generated from the following file:

- [GameState.h](#)

6.11 testOutput Struct Reference

struct for object that is used for the YOLO tests.

```
#include <common.h>
```

Public Attributes

- `vector< stringstream > singleFile`
vector of stringstreams for single test output files
- `vector< stringstream > csvSingleFile`
vector of stringstreams for single test output csv files
- `stringstream fileString`
stringstream for the whole test output file
- `stringstream csvFileString`
stringstream for the whole test output csv file
- `int fileCounter`
counter for how many files are already written
- `int debugTick`
current tick (seconds) for test
- `int debugSingleOutput`
tick value for when a single output file should be generated
- `double allClasses`
adds up the munchkin classes found in a single frame to calculate the average class count
- `double inferenceTime`
adds up the inference time for each frame to calculate the average inference time
- `double allPropability`
double value to add up all class propabilities to calculate the average class propability
- `string YoloName`
name of yolo net that is tested
- `string MunchkinClassesNum`
Number of munchkin classes (cards) that are used in the test.

6.11.1 Detailed Description

struct for object that is used for the YOLO tests.

6.11.2 Member Data Documentation

6.11.2.1 `allClasses`

```
double testOutput::allClasses
```

adds up the munchkin classes found in a single frame to calculate the average class count

6.11.2.2 `allPropability`

```
double testOutput::allPropability
```

double value to add up all class propabilities to calculate the average class propability

6.11.2.3 csvFileString

```
stringstream testOutput::csvFileString
```

stringstream for the whole test output csv file

6.11.2.4 csvSingleFile

```
vector<stringstream> testOutput::csvSingleFile
```

vector of stringstreams for single test output csv files

6.11.2.5 debugSingleOutput

```
int testOutput::debugSingleOutput
```

tick value for when a single output file should be generated

6.11.2.6 debugTick

```
int testOutput::debugTick
```

current tick (seconds) for test

6.11.2.7 fileCounter

```
int testOutput::fileCounter
```

counter for how many files are already written

6.11.2.8 fileString

```
stringstream testOutput::fileString
```

stringstream for the whole test output file

6.11.2.9 inferenceTime

```
double testOutput::inferenceTime
```

adds up the inference time for each frame to calculate the average inference time

6.11.2.10 MunchkinClassesNum

```
string testOutput::MunchkinClassesNum
```

Number of munchkin classes (cards) that are used in the test.

6.11.2.11 singleFile

```
vector<stringstream> testOutput::singleFile
```

vector of stringstreams for single test output files

6.11.2.12 YoloName

```
string testOutput::YoloName
```

name of yolo net that is tested

The documentation for this struct was generated from the following file:

- [common.h](#)

Chapter 7

File Documentation

7.1 ArucoGenerator.cpp File Reference

```
#include <ArucoGenerator.h>
```

Functions

- void [createArucoMarkers](#) ()

function to generate Aruco Markers. Dictionary for Aruco Markers are choosen inside this function.

7.1.1 Function Documentation

7.1.1.1 createArucoMarkers()

```
void createArucoMarkers ( )
```

function to generate Aruco Markers. Dictionary for Aruco Markers are choosen inside this function.

7.2 ArucoGenerator.h File Reference

class to generate new Aruco Markers

```
#include <vector>
#include "opencv2/opencv.hpp"
#include "opencv2/aruco.hpp"
#include <string>
#include <sstream>
#include <iostream>
#include <fstream>
```

Functions

- void [createArucoMarkers](#) ()

function to generate Aruco Markers. Dictionary for Aruco Markers are choosen inside this function.

7.2.1 Detailed Description

class to generate new Aruco Markers

Author

Benjamin Lueben

In this class the Aruco Markers can be generated. By now 50 4x4 Aruco Markers are generated.

7.2.2 Function Documentation

7.2.2.1 [createArucoMarkers\(\)](#)

```
void createArucoMarkers ( )
```

function to generate Aruco Markers. Dictionary for Aruco Markers are choosen inside this function.

7.3 BadThings.cpp File Reference

```
#include <BadThings.h>
#include <MunchkinCards.h>
#include <GameState.h>
```

7.4 BadThings.h File Reference

Bad Things Functions for Munchkin Cards.

```
#include <functional>
```

Classes

- struct [BadThingsRetVal](#)

Namespaces

- [badthings](#)

Namespace containing all bad things functions. These can be assigned to a specific card.

Typedefs

- using [BadThingsFunc](#) = std::function< [BadThingsRetVal](#)([GameState](#) &, const [MunchkinCard](#) &)>
Callback type for bad things behaviour.

Functions

- [BadThingsRetVal](#) [badthings::emptyBadThings](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &[card](#))
Does nothing (placeholder)
- [BadThingsRetVal](#) [badthings::looseHand](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &[card](#))
Player loses 1 hand and the function checks whether or not he can still carry all his items that need hands.
- [BadThingsRetVal](#) [badthings::looseClass](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &[card](#))
Player loses his class.
- [BadThingsRetVal](#) [badthings::playerDies](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &[card](#))
Player dies.
- [BadThingsRetVal](#) [badthings::looseLevel](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &[card](#))
Player loses 1 level.
- [BadThingsRetVal](#) [badthings::maleDeadFemaleLevelDown](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &[card](#))
Male Players die. Female players loose 1 level.

7.4.1 Detailed Description

Bad Things Functions for Munchkin Cards.

Author

Benjamin Lueben

In this class all the BadThings functions for every Munchkin Card are defined.

7.4.2 Typedef Documentation

7.4.2.1 BadThingsFunc

```
using BadThingsFunc = std::function<BadThingsRetVal(GameState&, const MunchkinCard&)>
```

Callback type for bad things behaviour.

7.5 Button.cpp File Reference

```
#include "Button.h"
```

7.6 Button.h File Reference

Class to define buttons which are used for user input.

```
#include "opencv2/opencv.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include <functional>
```

Classes

- class [Button](#)
class for Buttons

Typedefs

- using [button_callback](#) = std::function< void(const [Button](#) &)>

Enumerations

- enum class [ButtonOrigin](#) { [topleft](#) , [topright](#) , [bottomleft](#) , [bottomright](#) }
specifies in what corner the button should be shown

7.6.1 Detailed Description

Class to define buttons which are used for user input.

Author

Benjamin Lueben

With this class different buttons can be defined which can be used for user input. Buttons are used when the user needs to confirm an action or to exit the tutorial.

7.6.2 Typedef Documentation

7.6.2.1 button_callback

```
using button_callback = std::function<void(const Button&)>
```

7.6.3 Enumeration Type Documentation

7.6.3.1 ButtonOrigin

```
enum ButtonOrigin [strong]
```

specifies in what corner the button should be shown

Enumerator

topleft	top left corner
topright	top right corner
bottomleft	bottom left corner
bottomright	bottom right corner

7.7 CameraCalibration.cpp File Reference

```
#include <CameraCalibration.h>
```

Functions

- void [createKnownBoardPositions](#) (Size boardSize, float squareEdgeLength, vector< Point3f > &corners)
creates known boards position for chessboard used to calibrate camera.
- void [getChessboardCorners](#) (vector< Mat > images, vector< vector< Point2f >> &allFoundCorners, bool showResults)
function to get chessboard corners from given images.
- bool [saveCameraCalibration](#) (string name, Mat cameraMatrix, Mat distanceCoefficients)
function to save camera calibration to calibration file.
- bool [loadCameraCalibration](#) (string name, Mat &cameraMatrix, Mat &distanceCoefficients)
function so load camera calibration from file
- void [cameraCalibration](#) (vector< Mat > calibrationImages, Size boardSize, float squareEdgeLength, Mat &cameraMatrix, Mat &distanceCoefficients)
camera calibration function to process the actual camera calibration.
- void [cameraCalibrationProcess](#) (Mat &cameraMatrix, Mat &distanceCoefficients, Size [chessboardDimensions](#), float [calibrationSquareDimension](#))
function to guide through the camera calibration process. Can take Pictures of chess board to calibrate camera with

7.7.1 Function Documentation

7.7.1.1 cameraCalibration()

```
void cameraCalibration (
    vector< Mat > calibrationImages,
    Size boardSize,
    float squareEdgeLength,
    Mat & cameraMatrix,
    Mat & distanceCoefficients )
```

camera calibration function to process the actual camera calibration.

Parameters

<i>calibrationImages</i>	vector of mat objects that store the calibration images.
<i>boardSize</i>	size of chess board used to calibrate camera e.g 9x6
<i>squareEdgeLength</i>	edge length of single square tile on calibration board
<i>cameraMatrix</i>	reference of mat object for camera matrix to be stored in
<i>distanceCoefficients</i>	reference of mat object for distance coefficients to be stored in

7.7.1.2 cameraCalibrationProcess()

```
void cameraCalibrationProcess (
    Mat & cameraMatrix,
    Mat & distanceCoefficients,
    Size chessboardDimensions,
    float calibrationSquareDimension )
```

function to guide through the camera calibration process. Can take Pictures of chess board to calibrate camera with

Parameters

<i>cameraMatrix</i>	reference of mat object to store camera matrix in after calibration.
<i>distanceCoefficients</i>	reference of mat object to store distance coefficients in after calibration.
<i>chessboardDimensions</i>	size of chessboard dimensions e.g 9x6
<i>calibrationSquareDimensions</i>	edge length of single square tile on calibration board

7.7.1.3 createKnownBoardPositions()

```
void createKnownBoardPositions (
    Size boardSize,
```



```
float squareEdgeLength,
vector< Point3f > & corners )
```

creates known boards position for chessboard used to calibrate camera.

Parameters

<i>boardSize</i>	size of the calibration chess board. e.g 9x6
<i>squareEdgeLength</i>	Square edge length of single calibration board tile. Needs to be measured from printed chess board.
<i>corners</i>	reference of chessboard corners that are returned.

7.7.1.4 getChessboardCorners()

```
void getChessboardCorners (
    vector< Mat > images,
    vector< vector< Point2f >> & allFoundCorners,
    bool showResult = false )
```

function to get chessboard corners from given images.

Parameters

<i>images</i>	Vector of Mat objects with calibration pictures of chessboard.
<i>allFoundCorners</i>	vector with vector of Points that return all found corners of tiles inside the chessboard pictures.
<i>showResult</i>	bool of whether or not the results should be shown on the output image.

7.7.1.5 loadCameraCalibration()

```
bool loadCameraCalibration (
    string name,
    Mat & cameraMatrix,
    Mat & distanceCoefficients )
```

function so load camera calibration from file

Parameters

<i>name</i>	filepath for camera calibration file
<i>cameraMatrix</i>	reference to mat object to store camera matrix in and return camera matrix.
<i>distanceCoefficients</i>	reference to mat object to store distance coefficients in and return them.

Returns

true => file could be opened and camera matrix was loaded; false => file could not be opened.

7.7.1.6 saveCameraCalibration()

```
bool saveCameraCalibration (
    string name,
    Mat cameraMatrix,
    Mat distanceCoefficients )
```

function to save camera calibration to calibration file.

Parameters

<i>name</i>	name for the filepath to store camera matrix in
<i>cameraMatrix</i>	calculated camera matrix mat object to store inside calibration file
<i>distanceCoefficients</i>	distance coefficients that are also stored within the camera calibration file

Returns

true => file was saved successfully; false => file could not be created;

7.8 CameraCalibration.h File Reference

Camera Calibration with OpenCV Chessboard.

```
#include "opencv2/opencv.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/aruco.hpp"
#include "opencv2/calib3d.hpp"
#include "opencv2/video.hpp"
#include "opencv2/videoio.hpp"
```

Functions

- void [createKnownBoardPositions](#) (Size boardSize, float squareEdgeLength, vector< Point3f > &corners)
creates known boards position for chessboard used to calibrate camera.
- void [getChessboardCorners](#) (vector< Mat > images, vector< vector< Point2f >> &allFoundCorners, bool showResult=false)
function to get chessboard corners from given images.
- bool [saveCameraCalibration](#) (string name, Mat cameraMatrix, Mat distanceCoefficients)
function to save camera calibration to calibration file.
- bool [loadCameraCalibration](#) (string name, Mat &cameraMatrix, Mat &distanceCoefficients)
function so load camera calibration from file
- void [cameraCalibration](#) (vector< Mat > calibrationImages, Size boardSize, float squareEdgeLength, Mat &cameraMatrix, Mat &distanceCoefficients)
camera calibration function to process the actual camera calibration.
- void [cameraCalibrationProcess](#) (Mat &cameraMatrix, Mat &distanceCoefficients, Size [chessboardDimensions](#), float [calibrationSquareDimension](#))
function to guide through the camera calibration process. Can take Pictures of chess board to calibrate camera with

7.8.1 Detailed Description

Camera Calibration with OpenCV Chessboard.

Author

Benjamin Lueben

In this class the Webcam gets calibrated for use of Aruco Markers. Calibration is done with OpenCV and Chessboard pattern. This Class is based on a series of youtube tutorials from George Lecakes regarding OpenCV Camera Calibration and Aruco Markers (pt1 of Camera Calibration: https://www.youtube.com/watch?v=HNFpbw-1e_w)

7.8.2 Function Documentation

7.8.2.1 cameraCalibration()

```
void cameraCalibration (
    vector< Mat > calibrationImages,
    Size boardSize,
    float squareEdgeLength,
    Mat & cameraMatrix,
    Mat & distanceCoefficients )
```

camera calibration function to process the actual camera calibration.

Parameters

<i>calibrationImages</i>	vector of mat objects that store the calibration images.
<i>boardSize</i>	size of chess board used to calibrate camera e.g 9x6
<i>squareEdgeLength</i>	edge length of single square tile on calibration board
<i>cameraMatrix</i>	reference of mat object for camera matrix to be stored in
<i>distanceCoefficients</i>	reference of mat object for distance coefficients to be stored in

7.8.2.2 cameraCalibrationProcess()

```
void cameraCalibrationProcess (
    Mat & cameraMatrix,
    Mat & distanceCoefficients,
    Size chessboardDimensions,
    float calbirationSquareDimension )
```

function to guide through the camera calibration process. Can take Pictures of chess board to calibrate camera with

Parameters

<i>cameraMatrix</i>	reference of mat object to store camera matrix in after calibration.
<i>distanceCoefficients</i>	reference of mat object to store distance coefficients in after calibration.
<i>chessboardDimensions</i>	size of chessboard dimensions e.g 9x6
<i>calibrationSquareDimensions</i>	edge length of single square tile on calibration board

7.8.2.3 createKnownBoardPositions()

```
void createKnownBoardPositions (
    Size boardSize,
    float squareEdgeLength,
    vector< Point3f > & corners )
```

creates known boards position for chessboard used to calibrate camera.

Parameters

<i>boardSize</i>	size of the calibration chess board. e.g 9x6
<i>squareEdgeLength</i>	Square edge length of single calibration board tile. Needs to be measured from printed chess board.
<i>corners</i>	reference of chessboard corners that are returned.

7.8.2.4 getChessboardCorners()

```
void getChessboardCorners (
    vector< Mat > images,
    vector< vector< Point2f >> & allFoundCorners,
    bool showResult = false )
```

function to get chessboard corners from given images.

Parameters

<i>images</i>	Vector of Mat objects with calibration pictures of chessboard.
<i>allFoundCorners</i>	vector with vector of Points that return all found corners of tiles inside the chessboard pictures.
<i>showResult</i>	bool of wether or not the results should be shown on the output image.

7.8.2.5 loadCameraCalibration()

```
bool loadCameraCalibration (
    string name,
```

```
Mat & cameraMatrix,
Mat & distanceCoefficients )
```

function so load camera calibration from file

Parameters

<i>name</i>	filepath for camera calibration file
<i>cameraMatrix</i>	reference to mat object to store camera matrix in and return camera matrix.
<i>distanceCoefficients</i>	reference to mat object to store distance coefficients in and return them.

Returns

true => file could be opened and camera matrix was loaded; false => file could not be opened.

7.8.2.6 saveCameraCalibration()

```
bool saveCameraCalibration (
    string name,
    Mat cameraMatrix,
    Mat distanceCoefficients )
```

function to save camera calibration to calibration file.

Parameters

<i>name</i>	name for the filepath to store camera matrix in
<i>cameraMatrix</i>	calculated camera matrix mat object to store inside calibration file
<i>distanceCoefficients</i>	distance coefficients that are also stored within the camera calibration file

Returns

true => file was saved successfully; false => file could not be created;

7.9 CardTypeActions.cpp File Reference

```
#include "CardTypeActions.h"
#include <iostream>
#include "GameState.h"
#include "MunchkinCards.h"
```

Functions

- bool [runAway](#) ([GameState](#) &gamestate)

7.9.1 Function Documentation

7.9.1.1 runAway()

```
bool runAway (
    GameState & gamestate )
```

7.10 CardTypeActions.h File Reference

Class to define the functions for munchkin card types.

```
#include <functional>
```

Classes

- struct [CardTypeRetVal](#)

Namespaces

- [cardtypeaction](#)
Namespace containing all cardtype functions. These are mapped to the actual card type.

Typedefs

- using [CardTypeFunc](#) = std::function< [CardTypeRetVal](#)([GameState](#) &, const [MunchkinCard](#) &)>

Functions

- [CardTypeRetVal](#) [cardtypeaction::curse](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &card)
processes card type curse
- [CardTypeRetVal](#) [cardtypeaction::joker](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &card)
processes card type joker
- [CardTypeRetVal](#) [cardtypeaction::monster](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &card)
processes card type monster
- [CardTypeRetVal](#) [cardtypeaction::munchClass](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &card)
processes card Type munchkin class
- [CardTypeRetVal](#) [cardtypeaction::race](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &card)
processes card type munchkin race
- [CardTypeRetVal](#) [cardtypeaction::item](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &card)
processes card type item
- [CardTypeRetVal](#) [cardtypeaction::itemBuff](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &card)
processes card type item buff
- [CardTypeRetVal](#) [cardtypeaction::lvlUp](#) ([GameState](#) &[gamestate](#), const [MunchkinCard](#) &card)
processes card type level up

7.10.1 Detailed Description

Class to define the functions for munchkin card types.

Author

Benjamin Lueben

In this class all the functions for the different munchkin card types are defined which are used to implement the game logic of the munchkin cardgame.

7.10.2 Typedef Documentation

7.10.2.1 CardTypeFunc

```
using CardTypeFunc = std::function<CardTypeRetVal(GameState&, const MunchkinCard&)>
```

7.11 common.h File Reference

class for common used tools, objects, etc

```
#include <array>
#include <sstream>
```

Classes

- struct `testOutput`
struct for object that is used for the YOLO tests.

Namespaces

- `munch_tut`
namespace for munchkin tutorial

Typedefs

- using `munch_tut::Color` = `std::array< int, 3 >`

7.11.1 Detailed Description

class for common used tools, objects, etc

Author

Benjamin Lueben

In this class helpfull things like own namespaces or custom structs are defined that can be used throughout the whole classes.

7.12 DetectionVariants.cpp File Reference

```
#include <DetectionVariants.h>
```

Functions

- int [YOLOMatching](#) (Mat &frame, const vector< String > &classes, float [confThreshold](#), float [nmsThreshold](#), [GameState](#) &gamestate, const vector< [MunchkinCard](#) > &cards, int [modus](#), [testOutput](#) &fileContent)

Function to match Munchkincards via YOLO Neuronal Network. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencl>)

- int [ArucoMatching](#) (Mat &frame, [GameState](#) &gamestate, Ptr< aruco::Dictionary > markerDictionary, float arucoSquareDimensions, Mat cameraMatrix, Mat distanceCoefficients, vector< [MunchkinCard](#) > cards)

function to use Aruco Markers as workaround for matching method. Part of practical project.

7.12.1 Function Documentation

7.12.1.1 ArucoMatching()

```
int ArucoMatching (
    Mat & frame,
    GameState & gamestate,
    Ptr< aruco::Dictionary > markerDictionary,
    float arucoSquareDimensions,
    Mat cameraMatrix,
    Mat distanceCoefficients,
    vector< MunchkinCard > cards )
```

function to use Aruco Markers as workaround for matching method. Part of practical project.

Parameters

<i>frame</i>	current frame of webcam input
<i>gamestate</i>	gamestate object which stores all necessary data for tutorial
<i>markerDictionary</i>	aruco dictionary object of what aruco dictionary is used and needs to be detected
<i>arucoSquareDimensions</i>	dimension of aruco square that is printed onto the cards. Needs to be measured.
<i>cameraMatrix</i>	mat object of camera matrix which stores camera calibration needed for aruco markers to function
<i>distanceCoefficients</i>	mat object of camera matrix which stores distance coefficients needed for aruco markers to function

Returns

1 if nothing failed.

7.12.1.2 YOLOMatching()

```
int YOLOMatching (
    Mat & frame,
    const vector< String > & classes,
    float confThreshold,
    float nmsThreshold,
    GameState & gamestate,
    const vector< MunchkinCard > & cards,
    int modus,
    testOutput & fileContent )
```

Function to match Munchkincards via YOLO Neuronal Network. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv>)

Parameters

<i>frame</i>	current frame of webcam input
<i>classes</i>	vector of class names that can be found by YOLO net
<i>confThreshold</i>	threshold for confidence with which matches should be declared as valid/correct.
<i>nmsThreshold</i>	Threshold for non-maximum suppression method to remove overlapping bounding boxes
<i>gamestate</i>	gamestate object which stores all necessary data for tutorial
<i>cards</i>	vector of munchkin cards object in which all cards are stored that can be detected with the tutorial.
<i>modus</i>	modus for yolo matching (1 => live tutorial; 2 => testing with class name and confidence displayed; 4 => testing without class name and confidence displayed)
<i>fileContent</i>	object of testOutput to store all data needed for testing

Returns

1 if nothing failed.

7.13 DetectionVariants.h File Reference

multiple Detection Variants for Munchkin Tutorial

```
#include "opencv2/aruco.hpp"
#include "opencv2/dnn.hpp"
#include "GameState.h"
#include "MunchkinCards.h"
#include <PostProcessFunctions.h>
#include <common.h>
```

Functions

- int **YOLOMatching** (Mat &frame, const vector< String > &classes, float confThreshold, float nmsThreshold, GameState &gamestate, const vector< MunchkinCard > &cards, int modus, testOutput &fileContent)

Function to match Munchkincards via YOLO Neuronal Network. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-open-cv>)

- int **ArucoMatching** (Mat &frame, GameState &gamestate, Ptr< aruco::Dictionary > markerDictionary, float arucoSquareDimensions, Mat cameraMatrix, Mat distanceCoefficients, vector< MunchkinCard > cards)

function to use Aruco Markers as workaround for matching method. Part of practical project.

7.13.1 Detailed Description

multiple Detection Variants for Munchkin Tutorial

Author

Benjamin Lueben

In this class multiple Detection Variants are implemented to detect Munchkin Cards. Detection Variants contain Aruco Markers, Template Matching and neuronal net matching with YOLO.

7.13.2 Function Documentation

7.13.2.1 ArucoMatching()

```
int ArucoMatching (
    Mat & frame,
    GameState & gamestate,
    Ptr< aruco::Dictionary > markerDictionary,
    float arucoSquareDimensions,
    Mat cameraMatrix,
    Mat distanceCoefficients,
    vector< MunchkinCard > cards )
```

function to use Aruco Markers as workaround for matching method. Part of practical project.

Parameters

<i>frame</i>	current frame of webcam input
<i>gamestate</i>	gamestate object which stores all necessary data for tutorial
<i>markerDictionary</i>	aruco dictionary object of what aruco dictionary is used and needs to be detected
<i>arucoSquareDimensions</i>	dimension of aruco square that is printed onto the cards. Needs to be measured.
<i>cameraMatrix</i>	mat object of camera matrix which stores camera calibration needed for aruco markers to function
<i>distanceCoefficients</i>	mat object of camera matrix which stores distance coefficients needed for aruco markers to function
<i>cards</i>	vector of munchkin cards object in which all cards are stored that can be detected with the tutorial.

Returns

1 if nothing failed.

7.13.2.2 YOLOMatching()

```
int YOLOMatching (
    Mat & frame,
    const vector< String > & classes,
    float confThreshold,
    float nmsThreshold,
    GameState & gamestate,
    const vector< MunchkinCard > & cards,
    int modus,
    testOutput & fileContent )
```

Function to match Munchkincards via YOLO Neuronal Network. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv>)

Parameters

<i>frame</i>	current frame of webcam input
<i>classes</i>	vector of class names that can be found by YOLO net
<i>confThreshold</i>	threshold for confidence with which matches should be declared as valid/correct.
<i>nmsThreshold</i>	Threshold for non-maximum suppression method to remove overlapping bounding boxes
<i>gamestate</i>	gamestate object which stores all necessary data for tutorial
<i>cards</i>	vector of munchkin cards object in which all cards are stored that can be detected with the tutorial.
<i>modus</i>	modus for yolo matching (1 => live tutorial; 2 => testing with class name and confidence displayed; 4 => testing without class name and confidence displayed)
<i>fileContent</i>	object of testOutput to store all data needed for testing

Returns

1 if nothing failed.

7.14 Extras.cpp File Reference

```
#include "Extras.h"
#include "GameState.h"
#include "MunchkinCards.h"
```

7.15 Extras.h File Reference

```
#include <functional>
```

Classes

- struct [ExtrasRetVal](#)

Namespaces

- [extras](#)

Typedefs

- using [ExtrasFunc](#) = std::function< [ExtrasRetVal](#)([GameState](#) &, const [MunchkinCard](#) &)>

7.15.1 Typedef Documentation

7.15.1.1 ExtrasFunc

```
using ExtrasFunc = std::function<ExtrasRetVal(GameState&, const MunchkinCard&)>
```

7.16 GameState.h File Reference

[GameState](#) class to store all needed parameters for tutorial.

```
#include "opencv2/opencv.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include <string>
#include <vector>
#include "Button.h"
```

Classes

- struct [MouseParams](#)
Holds data of a mouse event.
- struct [NNresult](#)
struct that holds custom result from YOLO Neuronal Net
- struct [PlayerStats](#)
Holds data of player stats.
- struct [GameState](#)

Enumerations

- enum class [MouseEvent](#) { [lclick](#) , [rclick](#) , [move](#) , [none](#) }
Specifies the mouse event.

7.16.1 Detailed Description

[GameState](#) class to store all needed parameters for tutorial.

Author

Benjamin Lueben

In this class all the parameters that are needed for the tutorial are stored.

7.16.2 Enumeration Type Documentation

7.16.2.1 MouseEvent

```
enum MouseEvent [strong]
```

Specifies the mouse event.

Enumerator

lclick	left click
rclick	right click
move	mouse move
none	empty event

7.17 InputEvent.h File Reference

Class for all input events of the player through mouseclicks.

Classes

- struct [InputEvent](#)

Enumerations

- enum class [EventType](#) {
 [LmPress](#) , [LmRelease](#) , [RmPress](#) , [RmRelease](#) ,
 [KeyPress](#) , [KeyRelease](#) }
 enum of possible event types

7.17.1 Detailed Description

Class for all input events of the player through mouseclicks.

Author

Benjamin Lueben

7.17.2 Enumeration Type Documentation

7.17.2.1 EventType

```
enum EventType [strong]
```

enum of possible event types

Enumerator

LmPress	left mouse press
LmRelease	left mouse release
RmPress	right mouse press
RmRelease	right mouse release
KeyPress	keyboard key press
KeyRelease	keyboard key release

7.18 main.cpp File Reference

```
#include "opencv2/opencv.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/aruco.hpp"
#include "opencv2/calib3d.hpp"
#include "opencv2/video.hpp"
#include "opencv2/dnn.hpp"
#include "opencv2/videoio.hpp"
#include "GameState.h"
#include "MunchkinCards.h"
#include "CardTypeActions.h"
#include "common.h"
#include "Button.h"
#include "ArucoGenerator.h"
#include "CameraCalibration.h"
#include "DetectionVariants.h"
#include "YOLODataGenerator.h"
#include <sstream>
```

```

#include <iostream>
#include <fstream>
#include <string>
#include <stdexcept>
#include <unordered_map>
#include <functional>
#include <random>
#include <map>
#include <iomanip>
#include <cmath>
#include <type_traits>

```

Functions

- void [button_exit](#) (const [Button](#) &b)
- void [button_continue](#) (const [Button](#) &b)
- void [button_end_turn](#) (const [Button](#) &b)
- void [logic](#) ()
- void [callBackFunction](#) (int event, int x, int y, int flags, void *userdata)
- int [TestYOLOMatching](#) ()
- int [startWebcamMonitoring](#) (const [Mat](#) &cameraMatrix, const [Mat](#) &distanceCoefficients, float arucoSquare↵
Dimensions)
- int [testArucoMatching](#) (const [Mat](#) &cameraMatrix, const [Mat](#) &distanceCoefficients, float arucoSquare↵
Dimensions)
- int [main](#) (int argv, char **argc)

Variables

- const float [calbirationSquareDimension](#) = 0.025f
- const float [arucoSquareDimension](#) = 0.02f
- const [Size](#) [chessboardDimensions](#) = [Size](#)(6, 9)
- float [confThreshold](#) = 0.7
- float [nmsThreshold](#) = 0.4
- int [modus](#) = 2
- int [debugSingleOutput](#) = 5
- bool [multipleTests](#) = false
- string [YoloName](#) = "40-20k"
- string [MunchkinClassesNum](#) = "40"
- string [testNum](#) = "3"
- vector< [String](#) > [classes](#)
- vector< [MunchkinCard](#) > [cards](#)
- std::unordered_map< [CardType](#), [CardTypeFunc](#) > [functMap](#)
- [GameState](#) [gamestate](#)

7.18.1 Function Documentation

7.18.1.1 `button_continue()`

```
void button_continue (
    const Button & b )
```

7.18.1.2 `button_end_turn()`

```
void button_end_turn (
    const Button & b )
```

7.18.1.3 `button_exit()`

```
void button_exit (
    const Button & b )
```

7.18.1.4 `callBackFunction()`

```
void callBackFunction (
    int event,
    int x,
    int y,
    int flags,
    void * userdata )
```

7.18.1.5 `logic()`

```
void logic ( )
```

7.18.1.6 `main()`

```
int main (
    int argv,
    char ** argc )
```


7.18.1.7 startWebcamMonitoring()

```
int startWebcamMonitoring (
    const Mat & cameraMatrix,
    const Mat & distanceCoefficients,
    float arucoSquareDimensions )
```

7.18.1.8 testArucoMatching()

```
int testArucoMatching (
    const Mat & cameraMatrix,
    const Mat & distanceCoefficients,
    float arucoSquareDimensions )
```

7.18.1.9 TestYOLOMatching()

```
int TestYOLOMatching ( )
```

7.18.2 Variable Documentation

7.18.2.1 arucoSquareDimension

```
const float arucoSquareDimension = 0.02f
```

7.18.2.2 calbirationSquareDimension

```
const float calbirationSquareDimension = 0.025f
```

7.18.2.3 cards

```
vector<MunchkinCard> cards
```

7.18.2.4 chessboardDimensions

```
const Size chessboardDimensions = Size(6, 9)
```

7.18.2.5 classes

```
vector<String> classes
```

7.18.2.6 confThreshold

```
float confThreshold = 0.7
```

7.18.2.7 debugSingleOutput

```
int debugSingleOutput = 5
```

7.18.2.8 functMap

```
std::unordered_map<CardType, CardTypeFunc> functMap
```

7.18.2.9 gamestate

```
GameState gamestate
```

7.18.2.10 modus

```
int modus = 2
```

7.18.2.11 multipleTests

```
bool multipleTests = false
```

7.18.2.12 MunchkinClassesNum

```
string MunchkinClassesNum = "40"
```

7.18.2.13 nmsThreshold

```
float nmsThreshold = 0.4
```

7.18.2.14 testNum

```
string testNum = "3"
```

7.18.2.15 YoloName

```
string YoloName = "40-20k"
```

7.19 MunchkinCards.cpp File Reference

```
#include "MunchkinCards.h"
```

7.20 MunchkinCards.h File Reference

MunchkinCards class in which all parameters for the munchkin cards are stored.

```
#include <string>
#include <vector>
#include "opencv2/opencv.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include <functional>
#include "GameState.h"
#include "BadThings.h"
```

Classes

- class [MunchkinCard](#)

class of munchkin cards with params that could be needed for a single card

Enumerations

- enum class `ParentCardType` { `door` , `treasure` }
enum class for parent card types
- enum class `CardType` {
`curse` , `munchClass` , `joker` , `lvlUp` ,
`itemBuff` , `monster` , `race` , `item` ,
`removeCard` }
enum class for "child" card Types
- enum class `ItemType` {
`armor` , `shoes` , `hat` , `boni` ,
`weapon` , `joker` , `clothing` }
enum class of item types

7.20.1 Detailed Description

MunchkinCards class in which all parameters for the munchkin cards are stored.

Author

Benjamin Lueben

In this class the munchkin Card object is defined with every parameter the card could have and with custom constructors which could be used to define the cards more easily

7.20.2 Enumeration Type Documentation

7.20.2.1 CardType

```
enum CardType [strong]
```

enum class for "child" card Types

Enumerator

<code>curse</code>	card type curse
<code>munchClass</code>	card type munchkin class
<code>joker</code>	card type joker
<code>lvlUp</code>	card type level up
<code>itemBuff</code>	card type item buff
<code>monster</code>	card type monster
<code>race</code>	card type munchkin race
<code>item</code>	card type item
<code>removeCard</code>	param to signal removing a card

7.20.2.2 ItemType

```
enum ItemType [strong]
```

enum class of item types

Enumerator

armor	item type armor
shoes	item class shoes
hat	item class hat
boni	item class boni
weapon	item class weapon
joker	item class joker
clothing	item class clothing

7.20.2.3 ParentCardType

```
enum ParentCardType [strong]
```

enum class for parent card types

Enumerator

door	card type door
treasure	card type treasure

7.21 PostProcessFunctions.cpp File Reference

```
#include <PostProcessFunctions.h>
#include <algorithm>
#include <vector>
#include <cmath>
```

Functions

- **Color chooseCardColor** (int markerId, vector< MunchkinCard > cards)
function to choose color e.g for boundingBoxes that is bound to type of single munchkin card.
- vector< String > **getOutputsNames** (const cv::dnn::Net &net)
function to get Outputs names from neuronal net. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python/>)
- void **drawPred** (int classId, float conf, int left, int top, int right, int bottom, Mat &frame, vector< String > classes, vector< MunchkinCard > cards, int modus)

- function to draw prediction boxes for YOLO matching. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python/>)
- void `postprocess` (Mat &frame, const vector< Mat > &outs, float `confThreshold`, float `nmsThreshold`, GameState &gamestate, const vector< String > &classes, const vector< MunchkinCard > &cards, int `modus`, testOutput &fileContent)

function which handles the post processing of outputs of YOLO matching method. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python/>)

7.21.1 Function Documentation

7.21.1.1 chooseCardColor()

```
Color chooseCardColor (
    int markerId,
    vector< MunchkinCard > cards )
```

function to choose color e.g for boundingBoxes that is bound to type of single munchkin card.

Parameters

<i>markerId</i>	id of munchkin card that was detected by tutorial. (former showed marker id which was referenced with munchkin card)
<i>cards</i>	vector of munchkin card objects with all munchkin cards that are currently usable with the tutorial.

Returns

cv::Color object with color for card (BGR)

7.21.1.2 drawPred()

```
void drawPred (
    int classId,
    float conf,
    int left,
    int top,
    int right,
    int bottom,
    Mat & frame,
    vector< String > classes,
    vector< MunchkinCard > cards,
    int modus )
```

function to draw prediction boxes for YOLO matching. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python/>)

Parameters

<i>classId</i>	class id of found class
<i>conf</i>	confidence of class for predicted box
<i>left</i>	x value of top-left corner of bounding box to be drawn
<i>top</i>	width of bounding box to be drawn
<i>right</i>	y value of top-left corner of bounding box to be drawn
<i>bottom</i>	height of bounding box to be drawn
<i>frame</i>	Mat object of current frame of webcam input
<i>classes</i>	vector of classes name that can be found with YOLO net
<i>cards</i>	vector of MunchkinCards object which stores all current detectable munchkin cards
<i>modus</i>	modus for yolo matching (1 => live tutorial; 2 => testing with class name and confidence displayed; 4 => testing without class name and confidence displayed)

7.21.1.3 getOutputsNames()

```
vector<String> getOutputsNames (
    const cv::dnn::Net & net )
```

function to get Outputs names from neuronal net. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-py>)

Parameters

<i>net</i>	cv::dnn::Net reference of neuronal net that is used in tutorial.
------------	--

Returns

array of names that can be found by neuronal net.

7.21.1.4 postprocess()

```
void postprocess (
    Mat & frame,
    const vector< Mat > & outs,
    float confThreshold,
    float nmsThreshold,
    GameState & gamestate,
    const vector< String > & classes,
    const vector< MunchkinCard > & cards,
    int modus,
    testOutput & fileContent )
```

function which handles the post processing of outputs of YOLO matching method. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-py>)

Parameters

<i>frame</i>	Mat obejct of current frame of webcam input
<i>outs</i>	vector of mat objects in which outputs from YOLO net are stored in.
<i>confThreshold</i>	threshold for confidence with which matches should be declared as valid/correct.
<i>nmsThreshold</i>	Threshold for non-maximum suppression method to remove overlapping bounding boxes
<i>gamestate</i>	gamestate object which stores all necessary data for tutorial
<i>cards</i>	vector of munchkin cards object in which all cards are stored that can be detected with the tutorial.
<i>classes</i>	vector of classes name that can be found with YOLO net
<i>modus</i>	modus for yolo matching (1 => live tutorial; 2 => testing with class name and confidence displayed; 4 => testing withour class name and confidence displayed)
<i>fileContent</i>	object of testOutput to store all data needed for testing

7.22 PostProcessFunctions.h File Reference

all util/post process functions

```
#include <MunchkinCards.h>
#include <GameState.h>
#include <common.h>
```

Namespaces

- [constants](#)

Functions

- [Color chooseCardColor](#) (int markerId, vector< [MunchkinCard](#) > [cards](#))
function to choose color e.g for boundingBoxes that is bound to type of single munchkin card.
- vector< String > [getOutputsNames](#) (const cv::dnn::Net &net)
function to get Outputs names from neuronal net. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python/>)
- void [drawPred](#) (int classId, float conf, int left, int top, int right, int bottom, Mat &frame, vector< String > [classes](#), vector< [MunchkinCard](#) > [cards](#), int [modus](#))
function to draw prediction boxes for YOLO matching. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python/>)
- void [postprocess](#) (Mat &frame, const vector< Mat > &outs, float [confThreshold](#), float [nmsThreshold](#), [GameState](#) &[gamestate](#), const vector< String > &[classes](#), const vector< [MunchkinCard](#) > &[cards](#), int [modus](#), [testOutput](#) &[fileContent](#))
function which handles the post processing of outputs of YOLO matching method. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python/>)

Variables

- constexpr int [constants::yolo_grid_size](#) = 13

7.22.1 Detailed Description

all util/post process functions

Author

Benjamin Lueben

In this class all the needed util and post process functions for Detection Variants are defined.

7.22.2 Function Documentation

7.22.2.1 chooseCardColor()

```
Color chooseCardColor (
    int markerId,
    vector< MunchkinCard > cards )
```

function to choose color e.g for boundingBoxes that is bound to type of single munchkin card.

Parameters

<i>markerId</i>	id of munchkin card that was detected by tutorial. (former showed marker id which was referenced with munchkin card)
<i>cards</i>	vector of munchkin card objects with all munchkin cards that are currently usable with the tutorial.

Returns

cv::Color object with color for card (BGR)

7.22.2.2 drawPred()

```
void drawPred (
    int classId,
    float conf,
    int left,
    int top,
    int right,
    int bottom,
    Mat & frame,
    vector< String > classes,
    vector< MunchkinCard > cards,
    int modus )
```

function to draw prediction boxes for YOLO matching. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-py>)

Parameters

<i>classId</i>	class id of found class
<i>conf</i>	confidence of class for predicted box
<i>left</i>	x value of top-left corner of bounding box to be drawn
<i>top</i>	width of bounding box to be drawn
<i>right</i>	y value of top-left corner of bounding box to be drawn
<i>bottom</i>	height of bounding box to be drawn
<i>frame</i>	Mat object of current frame of webcam input
<i>classes</i>	vector of classes name that can be found with YOLO net
<i>cards</i>	vector of MunchkinCards object which stores all current detectable munchkin cards
<i>modus</i>	modus for yolo matching (1 => live tutorial; 2 => testing with class name and confidence displayed; 4 => testing without class name and confidence displayed)

7.22.2.3 getOutputsNames()

```
vector<String> getOutputsNames (
    const cv::dnn::Net & net )
```

function to get Outputs names from neuronal net. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-py>)

Parameters

<i>net</i>	cv::dnn::Net reference of neuronal net that is used in tutorial.
------------	--

Returns

array of names that can be found by neuronal net.

7.22.2.4 postprocess()

```
void postprocess (
    Mat & frame,
    const vector< Mat > & outs,
    float confThreshold,
    float nmsThreshold,
    GameState & gamestate,
    const vector< String > & classes,
    const vector< MunchkinCard > & cards,
    int modus,
    testOutput & fileContent )
```

function which handles the post processing of outputs of YOLO matching method. This function is based on online Tutorial by Sunita Nayak (<https://learnopencv.com/deep-learning-based-object-detection-using-yolo>)

Parameters

<i>frame</i>	Mat object of current frame of webcam input
<i>outs</i>	vector of mat objects in which outputs from YOLO net are stored in.
<i>confThreshold</i>	threshold for confidence with which matches should be declared as valid/correct.
<i>nmsThreshold</i>	Threshold for non-maximum suppression method to remove overlapping bounding boxes
<i>gamestate</i>	gamestate object which stores all necessary data for tutorial
<i>cards</i>	vector of munchkin cards object in which all cards are stored that can be detected with the tutorial.
<i>classes</i>	vector of classes name that can be found with YOLO net
<i>modus</i>	modus for yolo matching (1 => live tutorial; 2 => testing with class name and confidence displayed; 4 => testing without class name and confidence displayed)
<i>fileContent</i>	object of testOutput to store all data needed for testing

7.23 README.md File Reference

7.24 TTMunchkinTut.cpp File Reference

7.25 YOLODataGenerator.cpp File Reference

```
#include <YOLODataGenerator.h>
#include "opencv2/opencv.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
```

Functions

- `template<typename T>`
`T unirand (T a, T b)`
- `template<typename T>`
`T uninorm (T mean, T sdev)`
- `int generateDatasetNN ()`

generates datasets which were used to train YOLO nets. creates pictures and txt files with coordinates of bounding boxes relativ to picture size. values `num_images` and `num_inputPictures` are used for how many images and how many input pictures should be used. `num_inputPictures` means how many of loaded munchkin cards should be used.

7.25.1 Function Documentation

7.25.1.1 generateDatasetNN()

```
int generateDatasetNN ( )
```

generates datasets which were used to train YOLO nets. creates pictures and txt files with coordinates of bounding boxes relativ to picture size. values num_images and num_inputPictures are used for how many images and how many input pictures should be used. num_inputPictures means how many of loaded munchkin cards should be used.

Returns

1 when finished

7.25.1.2 uninorm()

```
template<typename T >
T uninorm (
    T mean,
    T sdev )
```

7.25.1.3 unirand()

```
template<typename T >
T unirand (
    T a,
    T b )
```

7.26 YOLODataGenerator.h File Reference

generates Data for YOLO training

```
#include <vector>
#include <sstream>
#include <iostream>
#include <fstream>
#include <string>
#include <stdexcept>
#include <unordered_map>
#include <functional>
#include <random>
#include <map>
#include <iomanip>
#include <cmath>
#include <type_traits>
```

Functions

- `template<typename T >`
`T unirand (T a, T b)`
- `template<typename T >`
`T uninorm (T mean, T sdev)`
- `int generateDatasetNN ()`

generates datasets which were used to train YOLO nets. creates pictures and txt files with coordinates of bounding boxes relativ to picture size. values num_images and num_inputPictures are used for how many images and how many input pictures should be used. num_inputPictures means how many of loaded munchkin cards should be used.

7.26.1 Detailed Description

generates Data for YOLO training

Author

Benjamin Lueben

In this class the functions needed for generating Data for YOLO neural net are defined.

7.26.2 Function Documentation

7.26.2.1 generateDatasetNN()

```
int generateDatasetNN ( )
```

generates datasets which were used to train YOLO nets. creates pictures and txt files with coordinates of bounding boxes relativ to picture size. values num_images and num_inputPictures are used for how many images and how many input pictures should be used. num_inputPictures means how many of loaded munchkin cards should be used.

Returns

1 when finished

7.26.2.2 uninorm()

```
template<typename T >
T uninorm (
    T mean,
    T sdev )
```

7.26.2.3 unirand()

```
template<typename T >
T unirand (
    T a,
    T b )
```


Index

- allClasses
 - testOutput, [38](#)
- allPropability
 - testOutput, [38](#)
- armor
 - MunchkinCards.h, [67](#)
- ArucoGenerator.cpp, [41](#)
 - createArucoMarkers, [41](#)
- ArucoGenerator.h, [41](#)
 - createArucoMarkers, [42](#)
- ArucoMatching
 - DetectionVariants.cpp, [54](#)
 - DetectionVariants.h, [56](#)
- arucoSquareDimension
 - main.cpp, [63](#)
- availableClasses
 - PlayerStats, [35](#)
- availableRaces
 - PlayerStats, [35](#)
- badThings
 - MunchkinCard, [30](#)
- badthings, [9](#)
 - emptyBadThings, [9](#)
 - looseClass, [10](#)
 - looseHand, [10](#)
 - looseLevel, [10](#)
 - maleDeadFemaleLevelDown, [11](#)
 - playerDies, [11](#)
- BadThings.cpp, [42](#)
- BadThings.h, [42](#)
 - BadThingsFunc, [43](#)
- BadThingsFunc
 - BadThings.h, [43](#)
- badThingsFunc
 - MunchkinCard, [30](#)
- BadThingsRetVal, [17](#)
- boni
 - MunchkinCards.h, [67](#)
- bonis
 - MunchkinCard, [30](#)
 - PlayerStats, [35](#)
- bottomleft
 - Button.h, [45](#)
- bottomright
 - Button.h, [45](#)
- Button, [17](#)
 - Button, [18](#), [19](#)
 - callback, [20](#)
 - color, [20](#)
 - draw, [19](#)
 - id, [20](#)
 - operator=, [19](#)
 - origin, [20](#)
 - poll_click, [19](#)
 - rect, [20](#)
 - visible, [21](#)
- Button.cpp, [44](#)
- Button.h, [44](#)
 - bottomleft, [45](#)
 - bottomright, [45](#)
 - button_callback, [44](#)
 - ButtonOrigin, [45](#)
 - toleft, [45](#)
 - topright, [45](#)
- button_callback
 - Button.h, [44](#)
- button_continue
 - main.cpp, [61](#)
- button_end_turn
 - main.cpp, [62](#)
- button_exit
 - main.cpp, [62](#)
- ButtonOrigin
 - Button.h, [45](#)
- buttons
 - GameState, [22](#)
- calibrationSquareDimension
 - main.cpp, [63](#)
- callback
 - Button, [20](#)
- callBackFunction
 - main.cpp, [62](#)
- cameraCalibration
 - CameraCalibration.cpp, [46](#)
 - CameraCalibration.h, [49](#)
- CameraCalibration.cpp, [45](#)
 - cameraCalibration, [46](#)
 - cameraCalibrationProcess, [46](#)
 - createKnownBoardPositions, [46](#)
 - getChessboardCorners, [47](#)
 - loadCameraCalibration, [47](#)
 - saveCameraCalibration, [48](#)
- CameraCalibration.h, [48](#)
 - cameraCalibration, [49](#)
 - cameraCalibrationProcess, [49](#)
 - createKnownBoardPositions, [50](#)
 - getChessboardCorners, [50](#)
 - loadCameraCalibration, [50](#)

- saveCameraCalibration, 51
- cameraCalibrationProcess
 - CameraCalibration.cpp, 46
 - CameraCalibration.h, 49
- canvas_size
 - GameState, 22
- cardName
 - MunchkinCard, 30
- cards
 - main.cpp, 63
- cardsConstr
 - MunchkinCard, 29
- CardType
 - MunchkinCards.h, 66
- cardtypeaction, 11
 - curse, 12
 - item, 12
 - itemBuff, 12
 - joker, 13
 - lvlUp, 13
 - monster, 13
 - munchClass, 14
 - race, 14
- CardTypeActions.cpp, 51
 - runAway, 52
- CardTypeActions.h, 52
 - CardTypeFunc, 53
- CardTypeFunc
 - CardTypeActions.h, 53
- CardTypeRetVal, 21
- carriesLargeItems
 - PlayerStats, 35
- chessboardDimensions
 - main.cpp, 63
- chooseCardColor
 - PostProcessFunctions.cpp, 68
 - PostProcessFunctions.h, 71
- classes
 - main.cpp, 64
- clickP
 - MouseParams, 25
- clothing
 - MunchkinCards.h, 67
- Color
 - munch_tut, 15
- color
 - Button, 20
- common.h, 53
- confThreshold
 - main.cpp, 64
- constants, 14
 - yolo_grid_size, 15
- createArucoMarkers
 - ArucoGenerator.cpp, 41
 - ArucoGenerator.h, 42
- createKnownBoardPositions
 - CameraCalibration.cpp, 46
 - CameraCalibration.h, 50
- csvFileString
 - testOutput, 38
- csvSingleFile
 - testOutput, 39
- curse
 - cardtypeaction, 12
 - MunchkinCards.h, 66
- curses
 - PlayerStats, 35
- debuff
 - MunchkinCard, 30
- debugSingleOutput
 - main.cpp, 64
 - testOutput, 39
- debugTick
 - testOutput, 39
- DetectionVariants.cpp, 54
 - ArucoMatching, 54
 - YOLOMatching, 55
- DetectionVariants.h, 55
 - ArucoMatching, 56
 - YOLOMatching, 57
- door
 - MunchkinCards.h, 67
- draw
 - Button, 19
- drawPred
 - PostProcessFunctions.cpp, 68
 - PostProcessFunctions.h, 71
- effect
 - MunchkinCard, 30
- emptyBadThings
 - badthings, 9
- end_turn
 - GameState, 22
- event
 - MouseParams, 25
- EventType
 - InputEvent.h, 60
- extras, 15
 - Extras.cpp, 57
 - Extras.h, 57
 - ExtrasFunc, 58
- ExtrasFunc
 - Extras.h, 58
- ExtrasRetVal, 21
- fileCounter
 - testOutput, 39
- fileString
 - testOutput, 39
- functMap
 - main.cpp, 64
- GameState, 21
 - buttons, 22
 - canvas_size, 22

- end_turn, 22
 - mouseparams, 23
 - nnResults, 23
 - player01, 23
 - remove_card, 23
 - run_away, 23
 - should_continue, 23
 - should_exit, 24
- gamestate
 - main.cpp, 64
- GameState.h, 58
 - lclick, 59
 - MouseEvent, 59
 - move, 59
 - none, 59
 - rclick, 59
- generateDatasetNN
 - YOLODataGenerator.cpp, 73
 - YOLODataGenerator.h, 75
- getChessboardCorners
 - CameraCalibration.cpp, 47
 - CameraCalibration.h, 50
- getOutputsNames
 - PostProcessFunctions.cpp, 69
 - PostProcessFunctions.h, 72
- hands
 - PlayerStats, 35
- handsNeeded
 - MunchkinCard, 31
- hasArmor
 - PlayerStats, 36
- hasHat
 - PlayerStats, 36
- hasShoes
 - PlayerStats, 36
- hat
 - MunchkinCards.h, 67
- id
 - Button, 20
 - NNresult, 33
- inferenceTime
 - testOutput, 39
- InputEvent, 24
 - type, 24
- InputEvent.h, 59
 - EventType, 60
 - KeyPress, 60
 - KeyRelease, 60
 - LmPress, 60
 - LmRelease, 60
 - RmPress, 60
 - RmRelease, 60
- item
 - cardtypeaction, 12
 - MunchkinCards.h, 66
- itemBuff
 - cardtypeaction, 12
- MunchkinCards.h, 66
- itemEffect
 - MunchkinCard, 31
- itemEffects
 - PlayerStats, 36
- itemLarge
 - MunchkinCard, 31
- itemNeeds
 - MunchkinCard, 31
- ItemType
 - MunchkinCards.h, 66
- itemType
 - MunchkinCard, 31
- itemValue
 - MunchkinCard, 31
- joker
 - cardtypeaction, 13
 - MunchkinCards.h, 66, 67
- KeyPress
 - InputEvent.h, 60
- KeyRelease
 - InputEvent.h, 60
- lclick
 - GameState.h, 59
- LmPress
 - InputEvent.h, 60
- LmRelease
 - InputEvent.h, 60
- loadCameraCalibration
 - CameraCalibration.cpp, 47
 - CameraCalibration.h, 50
- logic
 - main.cpp, 62
- looseClass
 - badthings, 10
- looseHand
 - badthings, 10
- looseLevel
 - badthings, 10
- lvi
 - PlayerStats, 36
- lviUp
 - cardtypeaction, 13
 - MunchkinCard, 32
 - MunchkinCards.h, 66
- main
 - main.cpp, 62
- main.cpp, 60
 - arucoSquareDimension, 63
 - button_continue, 61
 - button_end_turn, 62
 - button_exit, 62
 - calibrationSquareDimension, 63
 - callBackFunction, 62
 - cards, 63

- chessboardDimensions, 63
- classes, 64
- confThreshold, 64
- debugSingleOutput, 64
- functMap, 64
- gamestate, 64
- logic, 62
- main, 62
- modus, 64
- multipleTests, 64
- MunchkinClassesNum, 64
- nmsThreshold, 65
- startWebcamMonitoring, 62
- testArucoMatching, 63
- testNum, 65
- TestYOLOMatching, 63
- YoloName, 65
- maleDeadFemaleLevelDown
 - badthings, 11
- markerCorner
 - MouseParams, 25
- markerID
 - MunchkinCard, 32
- markerId
 - MouseParams, 25
- modus
 - main.cpp, 64
- monsStrength
 - MunchkinCard, 32
- monster
 - cardtypeaction, 13
 - MunchkinCards.h, 66
- MouseEvent
 - GameState.h, 59
- MouseParams, 24
 - clickP, 25
 - event, 25
 - markerCorner, 25
 - markerId, 25
 - poly, 26
 - tutText, 26
- mouseparams
 - GameState, 23
- move
 - GameState.h, 59
- multipleTests
 - main.cpp, 64
- munch_tut, 15
 - Color, 15
- munchClass
 - cardtypeaction, 14
 - MunchkinCards.h, 66
- munchClasses
 - PlayerStats, 36
- MunchkinCard, 26
 - badThings, 30
 - badThingsFunc, 30
 - bonis, 30
 - cardName, 30
 - cardsConstr, 29
 - debuff, 30
 - effect, 30
 - handsNeeded, 31
 - itemEffect, 31
 - itemLarge, 31
 - itemNeeds, 31
 - itemType, 31
 - itemValue, 31
 - lvlUp, 32
 - markerID, 32
 - monsStrength, 32
 - MunchkinCard, 28, 29
 - operator=, 29
 - parentCardType, 32
 - strengthBoni, 32
 - treasures, 32
 - type, 33
- MunchkinCards.cpp, 65
- MunchkinCards.h, 65
 - armor, 67
 - boni, 67
 - CardType, 66
 - clothing, 67
 - curse, 66
 - door, 67
 - hat, 67
 - item, 66
 - itemBuff, 66
 - ItemType, 66
 - joker, 66, 67
 - lvlUp, 66
 - monster, 66
 - munchClass, 66
 - ParentCardType, 67
 - race, 66
 - removeCard, 66
 - shoes, 67
 - treasure, 67
 - weapon, 67
- MunchkinClassesNum
 - main.cpp, 64
 - testOutput, 40
- munchRaces
 - PlayerStats, 37
- nmsThreshold
 - main.cpp, 65
- nnBBox
 - NNresult, 33
- NNresult, 33
 - id, 33
 - nnBBox, 33
- nnResults
 - GameState, 23
- none
 - GameState.h, 59

- operator=
 - Button, 19
 - MunchkinCard, 29
- origin
 - Button, 20
- ParentCardType
 - MunchkinCards.h, 67
- parentCardType
 - MunchkinCard, 32
- player01
 - GameState, 23
- playerDies
 - badthings, 11
- PlayerStats, 34
 - availableClasses, 35
 - availableRaces, 35
 - bonis, 35
 - carriesLargeltems, 35
 - curses, 35
 - hands, 35
 - hasArmor, 36
 - hasHat, 36
 - hasShoes, 36
 - itemEffects, 36
 - lvl, 36
 - munchClasses, 36
 - munchRaces, 37
 - runStrength, 37
 - sex, 37
 - strength, 37
- poll_click
 - Button, 19
- poly
 - MouseParams, 26
- postprocess
 - PostProcessFunctions.cpp, 69
 - PostProcessFunctions.h, 72
- PostProcessFunctions.cpp, 67
 - chooseCardColor, 68
 - drawPred, 68
 - getOutputsNames, 69
 - postprocess, 69
- PostProcessFunctions.h, 70
 - chooseCardColor, 71
 - drawPred, 71
 - getOutputsNames, 72
 - postprocess, 72
- race
 - cardtypeaction, 14
 - MunchkinCards.h, 66
- rclick
 - GameState.h, 59
- README.md, 73
- rect
 - Button, 20
- remove_card
 - GameState, 23
- removeCard
 - MunchkinCards.h, 66
- RmPress
 - InputEvent.h, 60
- RmRelease
 - InputEvent.h, 60
- run_away
 - GameState, 23
- runAway
 - CardTypeActions.cpp, 52
- runStrength
 - PlayerStats, 37
- saveCameraCalibration
 - CameraCalibration.cpp, 48
 - CameraCalibration.h, 51
- sex
 - PlayerStats, 37
- shoes
 - MunchkinCards.h, 67
- should_continue
 - GameState, 23
- should_exit
 - GameState, 24
- singleFile
 - testOutput, 40
- startWebcamMonitoring
 - main.cpp, 62
- strength
 - PlayerStats, 37
- strengthBoni
 - MunchkinCard, 32
- testArucoMatching
 - main.cpp, 63
- testNum
 - main.cpp, 65
- testOutput, 37
 - allClasses, 38
 - allPropability, 38
 - csvFileString, 38
 - csvSingleFile, 39
 - debugSingleOutput, 39
 - debugTick, 39
 - fileCounter, 39
 - fileString, 39
 - inferenceTime, 39
 - MunchkinClassesNum, 40
 - singleFile, 40
 - YoloName, 40
- TestYOLOMatching
 - main.cpp, 63
- toleft
 - Button.h, 45
- topright
 - Button.h, 45
- treasure
 - MunchkinCards.h, 67
- treasures

- MunchkinCard, [32](#)
- TTMunchkinTut.cpp, [73](#)
- tutText
 - MouseParams, [26](#)
- type
 - InputEvent, [24](#)
 - MunchkinCard, [33](#)
- uninorm
 - YOLODataGenerator.cpp, [74](#)
 - YOLODataGenerator.h, [75](#)
- unirand
 - YOLODataGenerator.cpp, [74](#)
 - YOLODataGenerator.h, [75](#)
- visible
 - Button, [21](#)
- weapon
 - MunchkinCards.h, [67](#)
- yolo_grid_size
 - constants, [15](#)
- YOLODataGenerator.cpp, [73](#)
 - generateDatasetNN, [73](#)
 - uninorm, [74](#)
 - unirand, [74](#)
- YOLODataGenerator.h, [74](#)
 - generateDatasetNN, [75](#)
 - uninorm, [75](#)
 - unirand, [75](#)
- YOLOMatching
 - DetectionVariants.cpp, [55](#)
 - DetectionVariants.h, [57](#)
- YoloName
 - main.cpp, [65](#)
 - testOutput, [40](#)