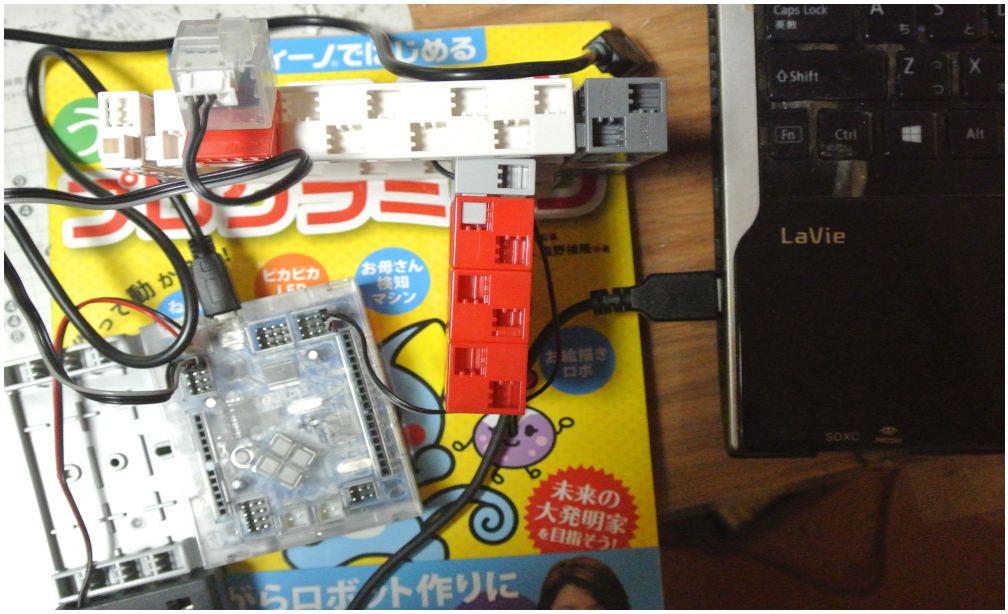
Scratch でテトリスとぷよぷよを作ってみよう

文責　：　高知大学名誉教授　中村　治

２０２０年から小学校でプログラミング教育が必修になるようです。フィンランド人が作った絵本 Linda Liukas 著「HELLO RUBY」（翻訳 :「ルビーの冒険」、RUBY はプログラミング言語の RUBY ではなく、主人公の女の子の名前です）が評判になっています。プログラミング言語を使わず、小さな子供たちにプログラミングの考え方を身に着けさせようという本です。米田昌悟著「プログラミング入門講座」SB Creative によると初心者にお勧めの学習サービスは「Hour of Code(Code Studio)」、「lightbot」、「CodeMonkey」、「Scratch」だそうです。「Hour of Code(Code Studio)」は無料で利用でき、ブロックを組み合わせてプログラミングします。昔、一時期流行っていましたが、今は忘れ去られていた Logo の復活です。子供に一流の道具を与えるだけでは上手くいかなかったので、今度はブロックを組み合わせてプログラミングするというメソッドと解決すべき問題を提示するためのパズルを組み合わせ、復活しました。「lightbot」は基本有料ですが、ブロックを組み合わせてプログラミングして、パズルを解きます。プロシージャの再帰呼び出しで繰り返しを実現しています。「CodeMonkey」も基本有料で、Logo とパズルを組み合わせたもので、ブロックではなく、文字列でプログラミングします。

「Scratch」は Logo にブロックを組み合わせてプログラミングするという糖衣を被せ、イベント駆動とオブジェクト指向をブラックボックス化したプログラミング言語です。タートルグラフィックス、スプライトの処理、オブジェクト指向、イベント駆動及び並列処理がブラックボックス化されサポートされています。昔、C や C++や Java でのゲーム作成の本を読んでも、猫を動かすだけでも大変で、自分でゲームを作ってみようとは思いませんでしたが、この大変だったスプライトの移動や衝突の判定や並列処理がブラックボックス化されていて、シューティングゲームやインベーダーゲームやブロック崩しが、作り方を書いた参考書やインターネットの記事がありますからそれらを見れば簡単に作れます。中植正剛、太田和志、鴨谷真知子著「Scratch で学ぶプログラミングとアルゴリズムの基本」日経 BP 社というコンピュータサイエンス指向の本も Logo で導入されたタートルグラフィックスもありますが、私の使ってみての印象としてはゲーム作成に特化したプログラミング言語だと感じます。欠点はプログラムが絵なので文字列である他のプログラミング言語のプログラムのように印刷が出来ないことやプログラムが 2 次元に広がっていてどこに何があるか探すのに時間がかかります。また子供たちが理解できるようにとの配慮から、データ構造や制御構造が貧弱なことです。Scratch2.0 も現在は、オンライン版とオフライン版があり、無料で使えます。オンライン版だと、他人が作ったゲームを探し、それで遊ぶだけになる危険性があるそうです。更に、「Viscuit」というちょっと遊んでみるには良い、面白い国産の言語もあります。インターネットで調べてください。

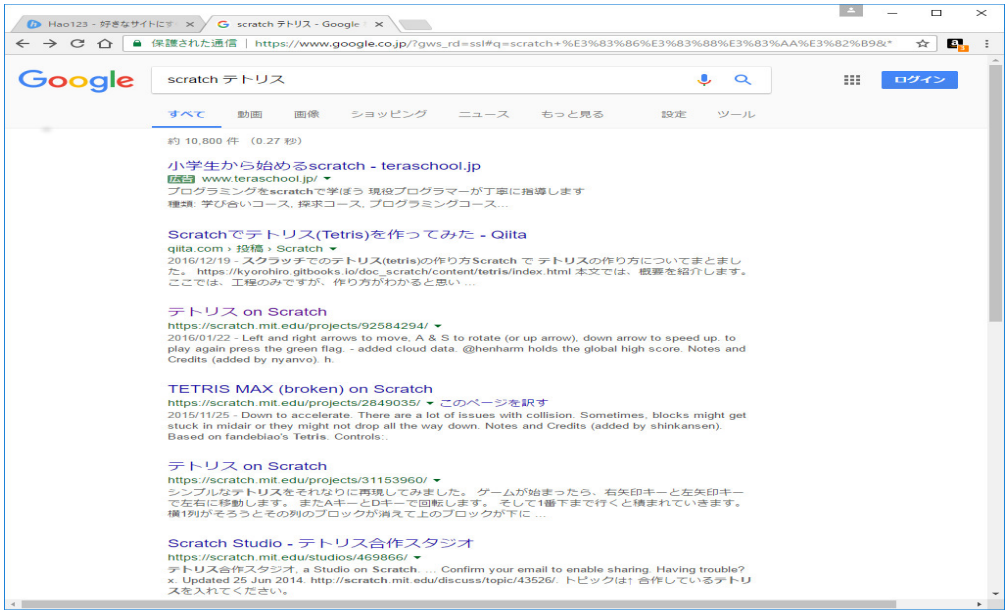
電子工作の分野では、アーチック社の「ロボティスト」等では Studuino ブロックプログラミング環境が用意されていて、サーボモーターの制御装置などがブラックボックス化されていて、アーチックブロックで組み立てた恐竜やロボットを制御することが出来、Raspberry PI や arduino で本格的な電子工作を始める前にやってみるといいです。最初に読むべき本は宇野泰正、塩野禎隆著「スタディーノではじめるうきうきロボットプログラミング」日経 BP 社だと思います。アーチックブロックで自動ドアを組み立て、サーボモーターがプログラミングで実際に動くと感動があります。１万数千円ぐらいお金はかかりますが是非やってみてください。人間、特に小さな子供達は、誰かが面白そうなことをやっていると、自分でもやって見たくなります。そうやって、自分の世界を広げていきます。ものを作るのは楽しいことです。



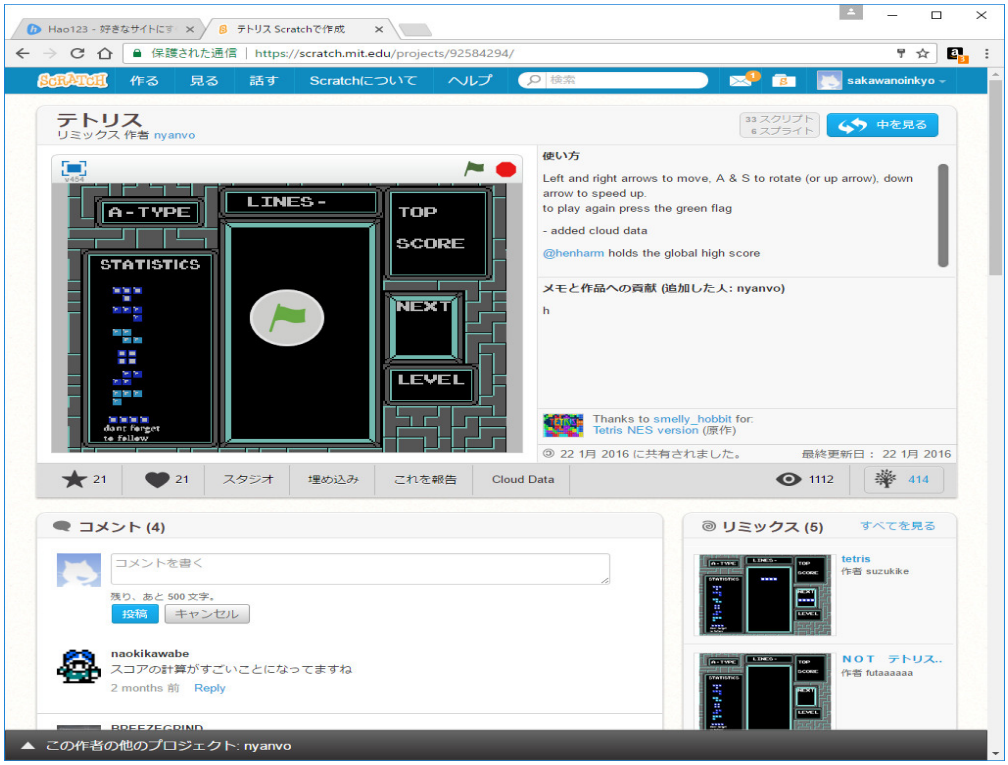
上図はアーチックブロックで自動ドアを組み立て、配線し、サーボモーターをプログラミングで動かしてみたところです。宇野泰正、塩野禎隆著「スタディーノではじめるうきうきロボットプログラミング」日経 BP 社では、このキットで出来るこの他のいろいろの工作も載っています。

私は持っていませんが、数万円出せば、人型ロボットがあります。先進的な小学校では、小学生たちが人型ロボットを制御するプログラミングに取り組んでいます。

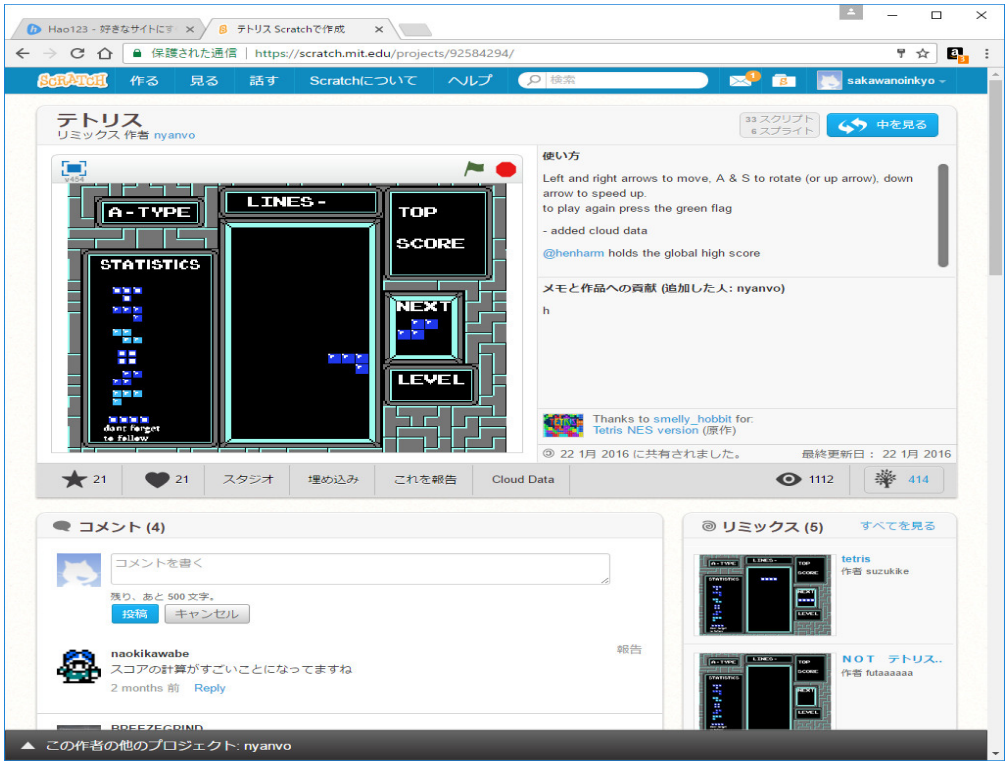
ここでは Scratch を取り上げます。Scratch の参考書やインターネットの情報も最近は増えてきています。それらの情報を見れば、「インベーダゲーム」や「ブロック崩し」までは、ゲームを楽しむことが出来るものが、下手は下手なりにですが、比較的容易に作ることが出来ます。しかし、「テトリス」や「ぷよぷよ」は少し厄介です。すでに、Scratch で作られた「テトリス」があります。Google で「Scratch　テトリス」で検索すると



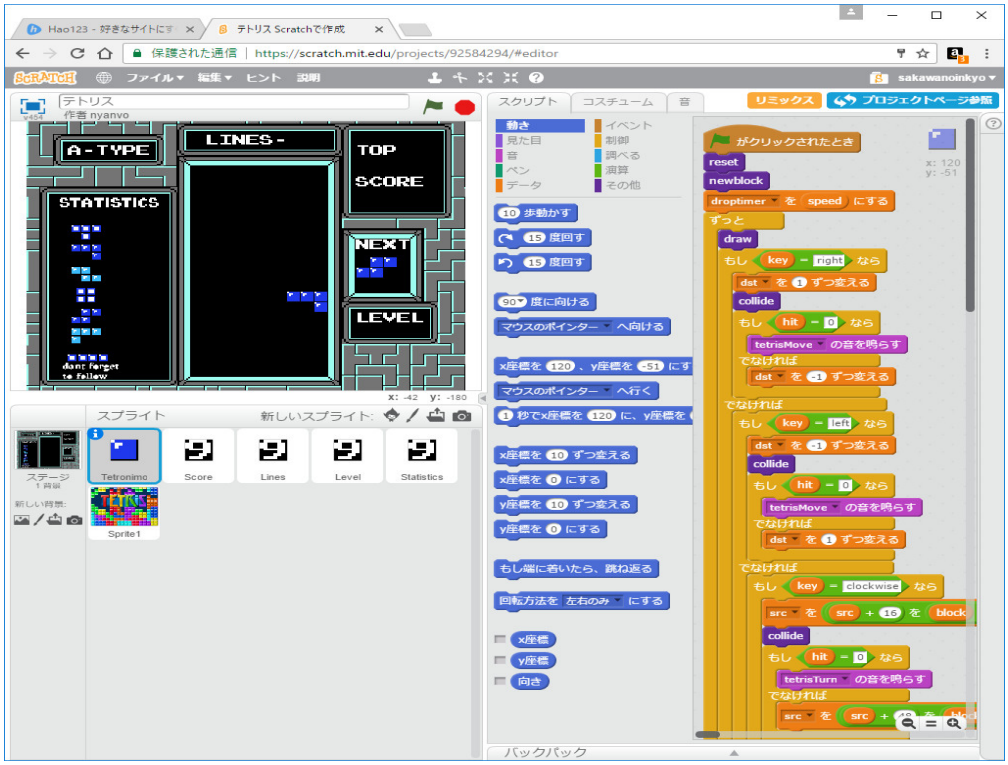
三番目の「テトリス on Scratch」をクリックすると



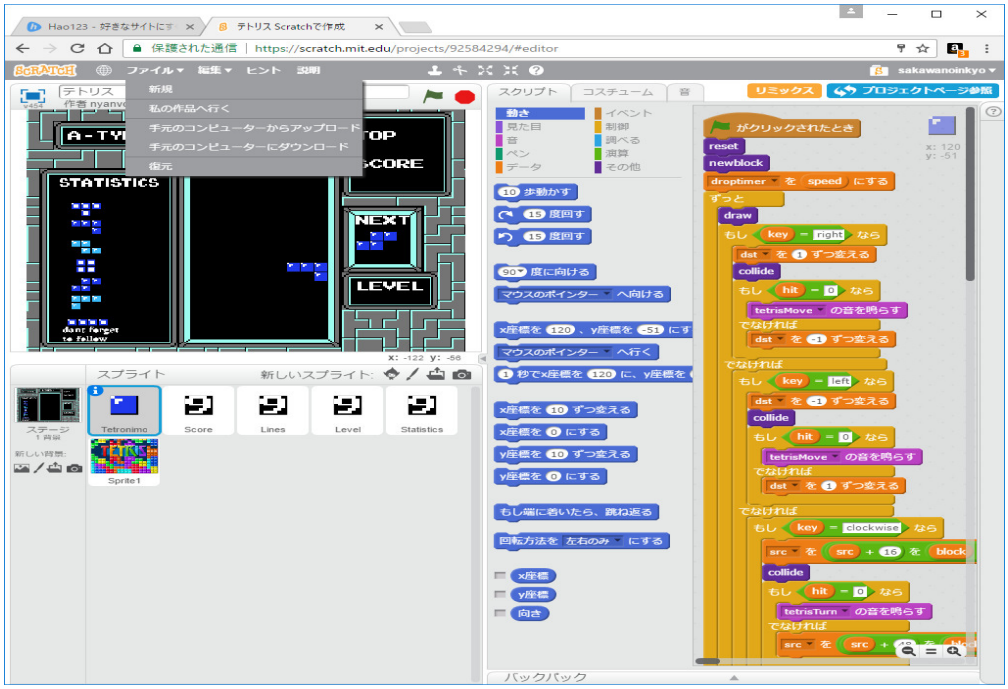
真ん中の旗をクリックするとテトリスが始まります。画面の右上隅の赤丸をクリックすると停止します。その隣の旗をクリックするとテトリスが再び始まります。



右上隅の「中を見る」をクリックすると



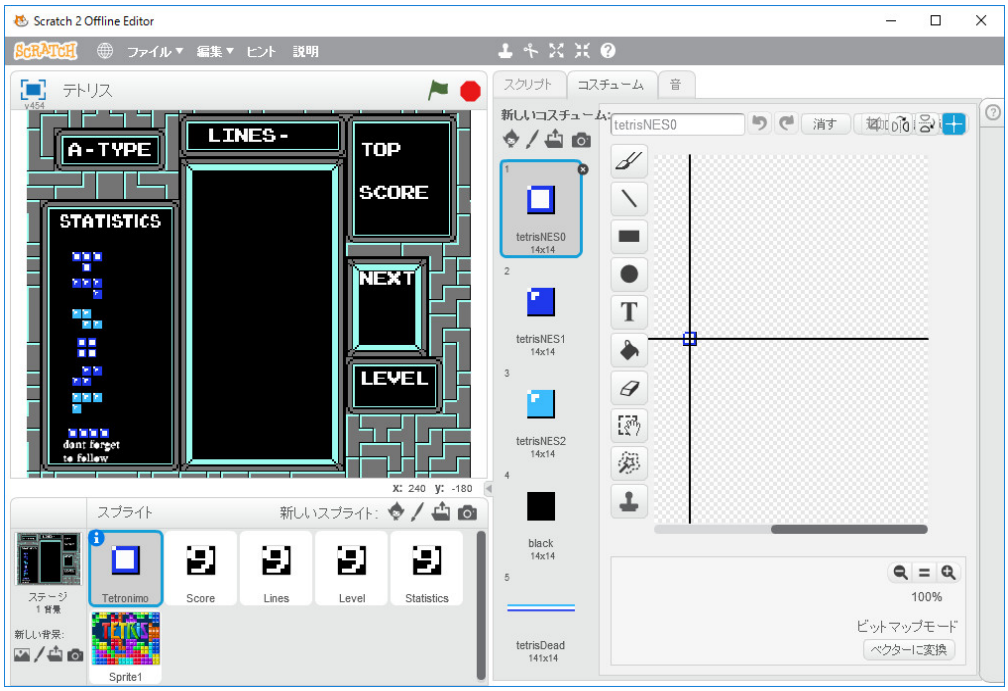
「ファイル」メニューの「手元のコンピュータにダウンロード」をクリックすると



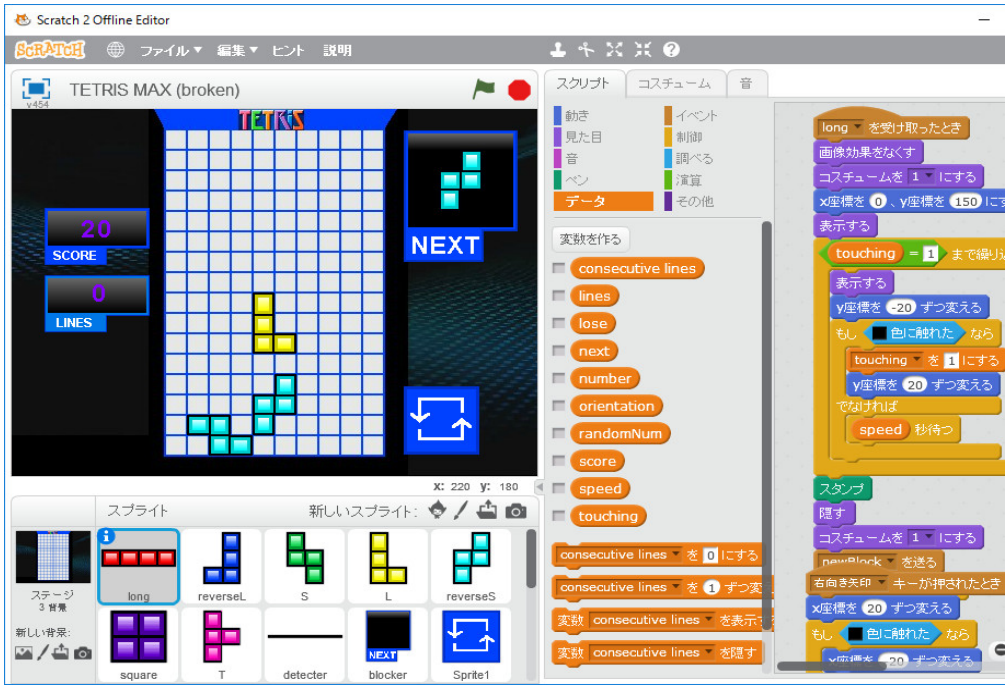
このプログラムを自分のパソコンに取り込むことが出来ます。

テトリスで遊ぶことが目的なら、これで目的を達したことになります。後は心ゆくまで楽しめばいいです。このようなプログラムを自分で作れるようになりたいと思ったら、一からこれと同じものを作ってみれば良いです。

プログラムを調べてみます。登場人物は背景とスプライトたちです。



スプライトは１４×１４の角の欠けた正方形を中央に配置し、いくつかのコスチュームがあります。背景は中央に７種類のテトリミノを落とす場となるフィールドが配置されていて、公式には縦２０行、横１０列とされているようです。このような背景を作者はどのようにして作ったのでしょう。見当もつきません。別のテトリスのプログラム TETRIS MAX(broken) の画面は



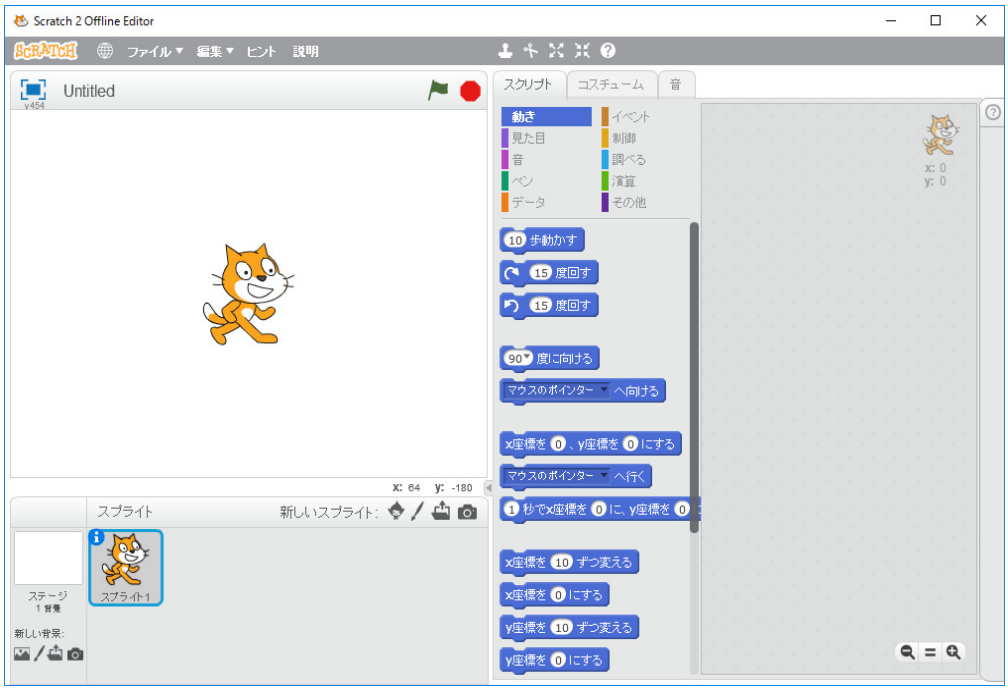
のようになっています。背景は単純ですが、見えているフィールドは縦１６行、横１０列です。ス

プライトは７種類のテトリミノです。コスチュームで回転の画像を持っています。「□色に触れた」の衝突判定を使っているので、上図のようなバグが発生しています。

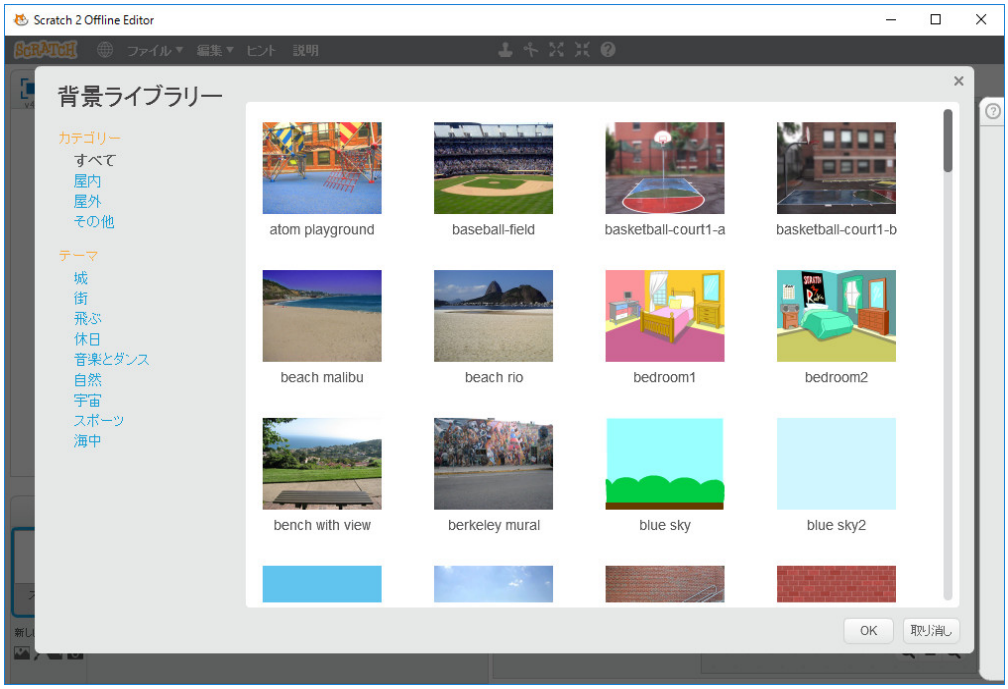
同じ背景を自力で作れないことが分かったので、同じものを作るのは断念して、単純なテトリスを作ることに目標を変えます。その時、参考になるのは平山尚著「プログラムはこうして作られる

プログラマの頭の中をのぞいてみよう」秀和システムです。４９６ページのこの本一冊を使ってテトリスの作り方、プログラミングの考え方を述べています。昔読んで、面白い本だと思ったのですが、内容をすっかり忘れていたので、今回もう一度読み直しました。そこまでやらなくてもと思うところもありますが、読む価値のある本です。フィールドは公式通り縦２０行、横１０列としましょう。平山尚さんの本のようにドットで正方形を作るのは大変だし、時間もかかるのでスプライトを使います。スプライトは正方形にするか、テトリミノのにするかの２通りの選択があります。コスチュームを使えば、回転の処理は単純になりますが、その正確なスプライトを自分で作れるかが問題です。スプライトの作り方を試してみましょう。

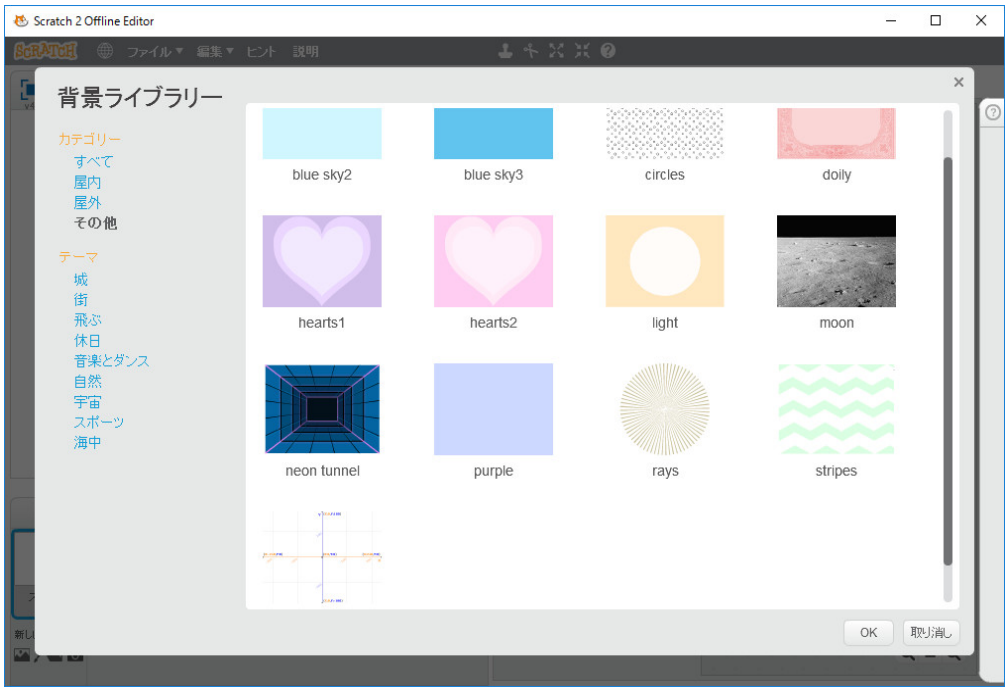
Scratch を立ち上げます。



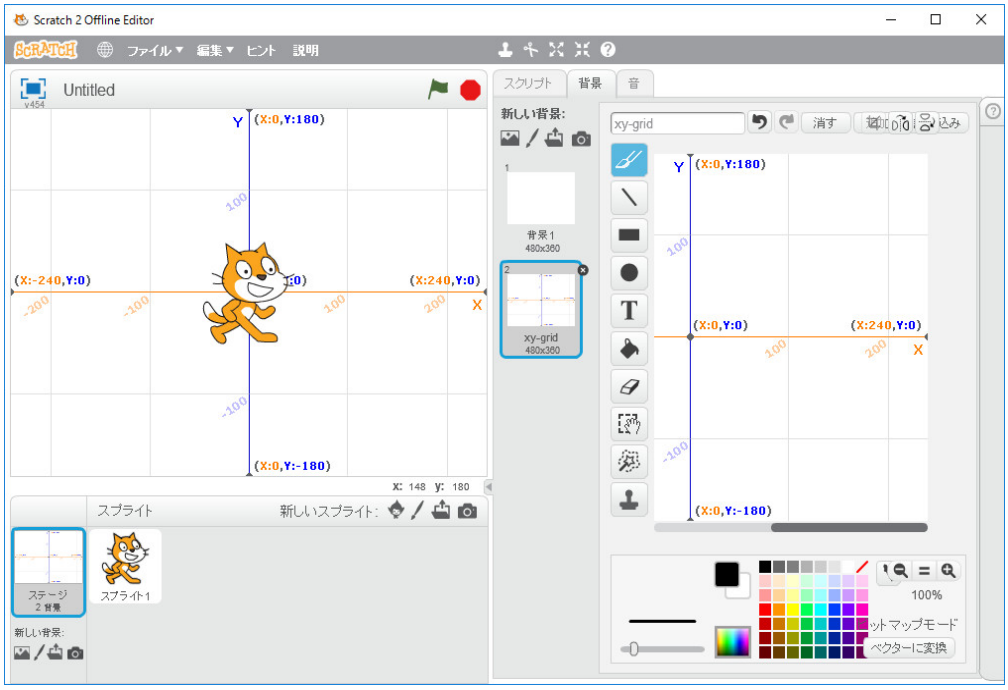
左下の「ステージ」をクリックして選択し、「新しい背景」の下の一番左のアイコンをクリックします。



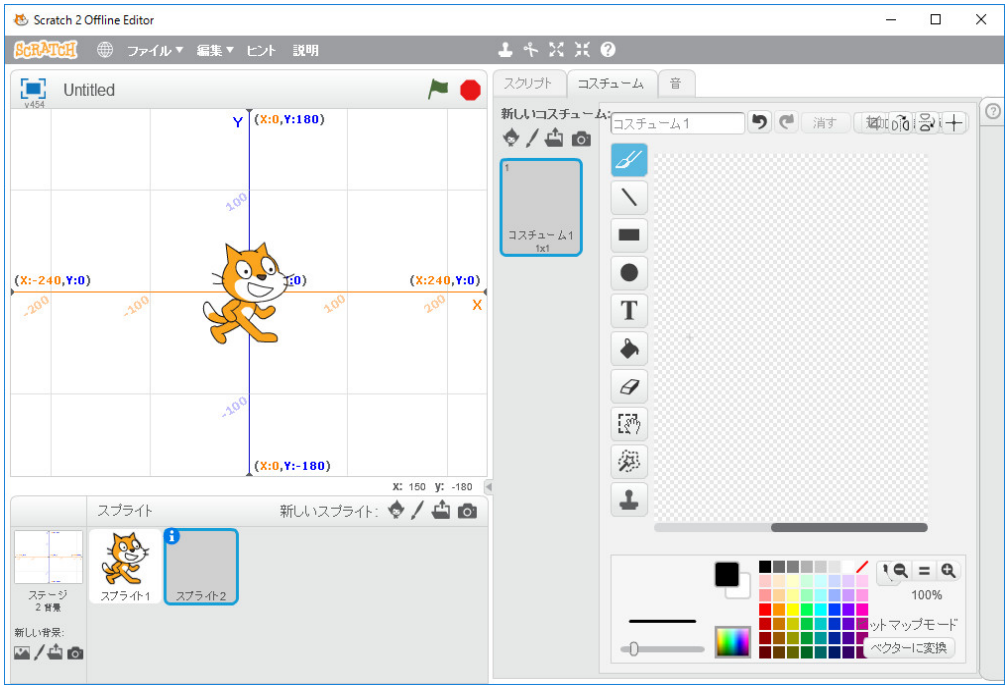
背景ライブラリーが現れます。「カテゴリー」の一番下のもの



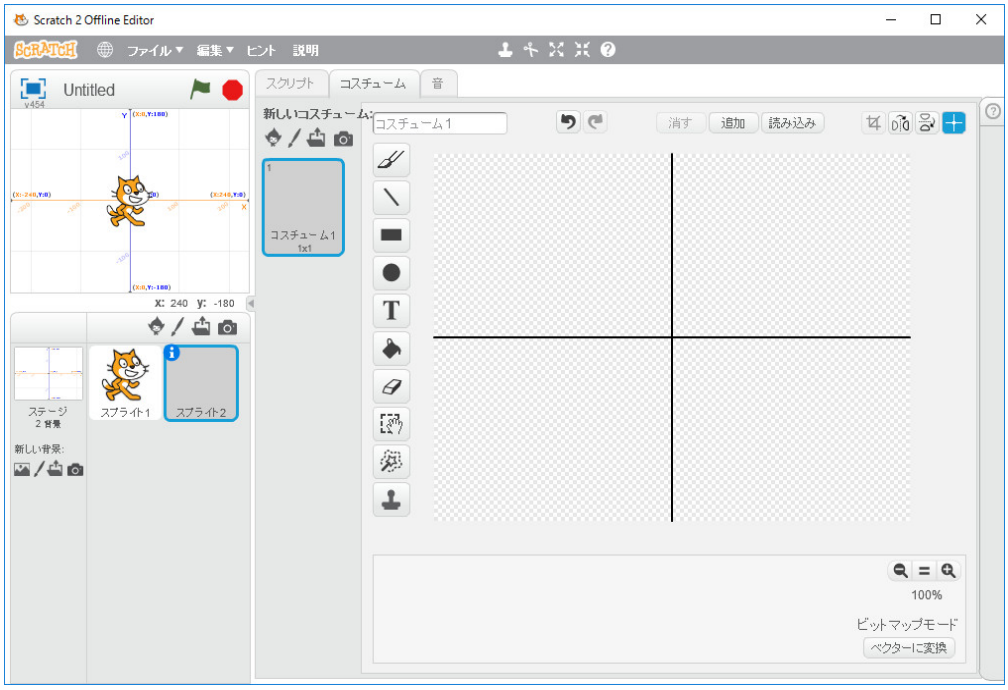
を選択します。



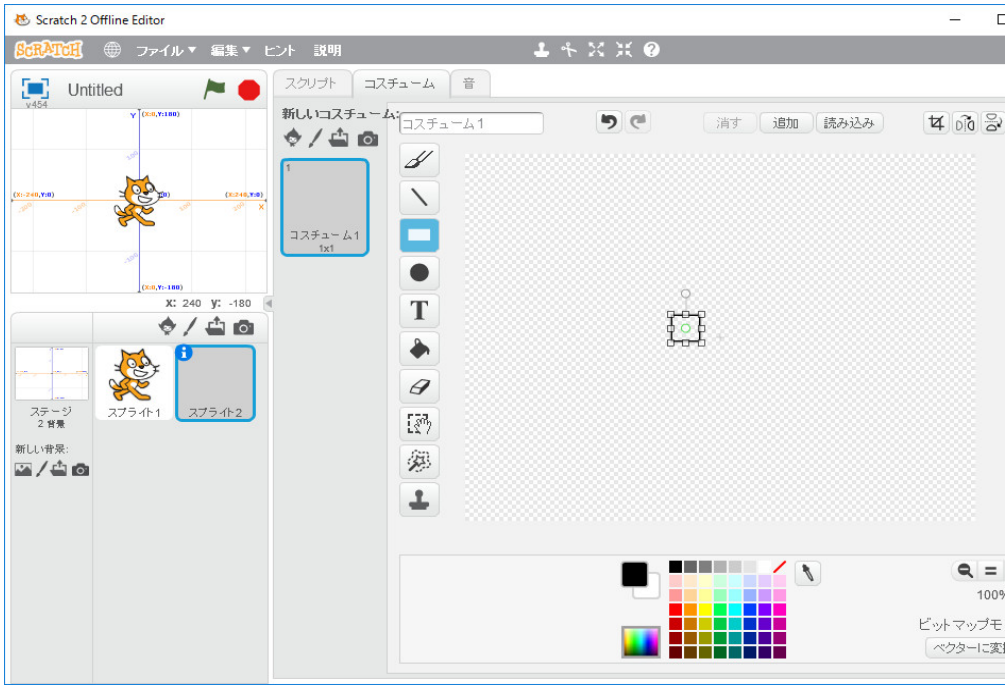
これが Scratch の座標系です。横４８０ドット、縦３６０ドットです。ここに枠を含めて縦２１行、横１２列のフィールドを取るには正方形を１４ドット×１４ドットが限界です。そこで、１４ドット×１４ドットの正方形のスプライトを作ってみましょう。「新しいスプライト」の左から２番目のペンのアイコンをクリックします。



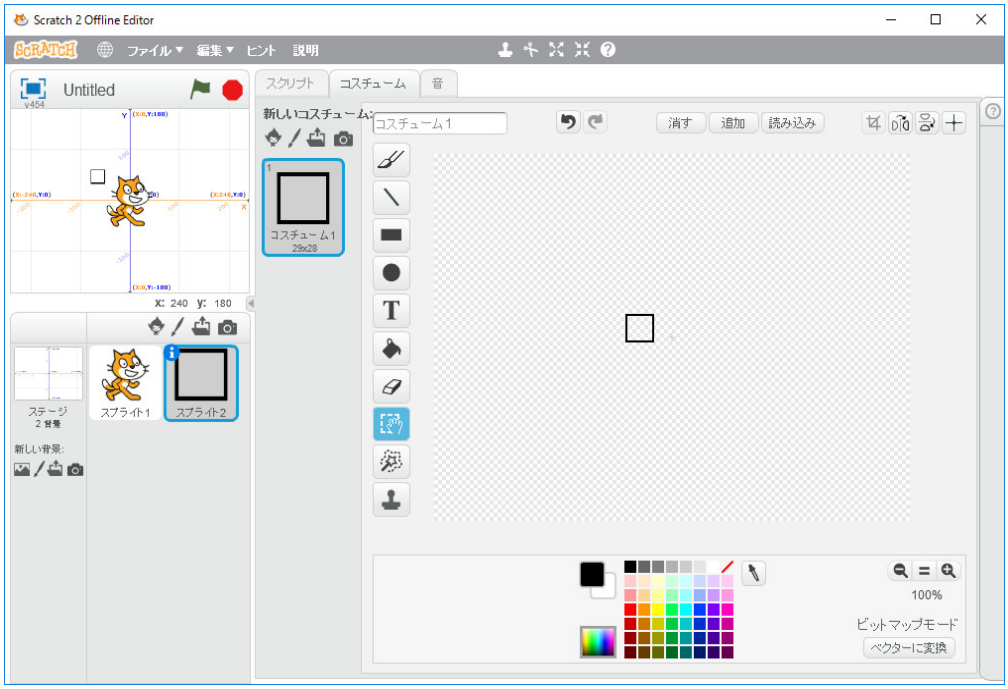
右の画面でスプライトを作ります。背景の画面の右下隅にある三角をクリックして、作図画面を大きくします。右上隅のプラスのアイコンをクリックすると中心線が表示されます。



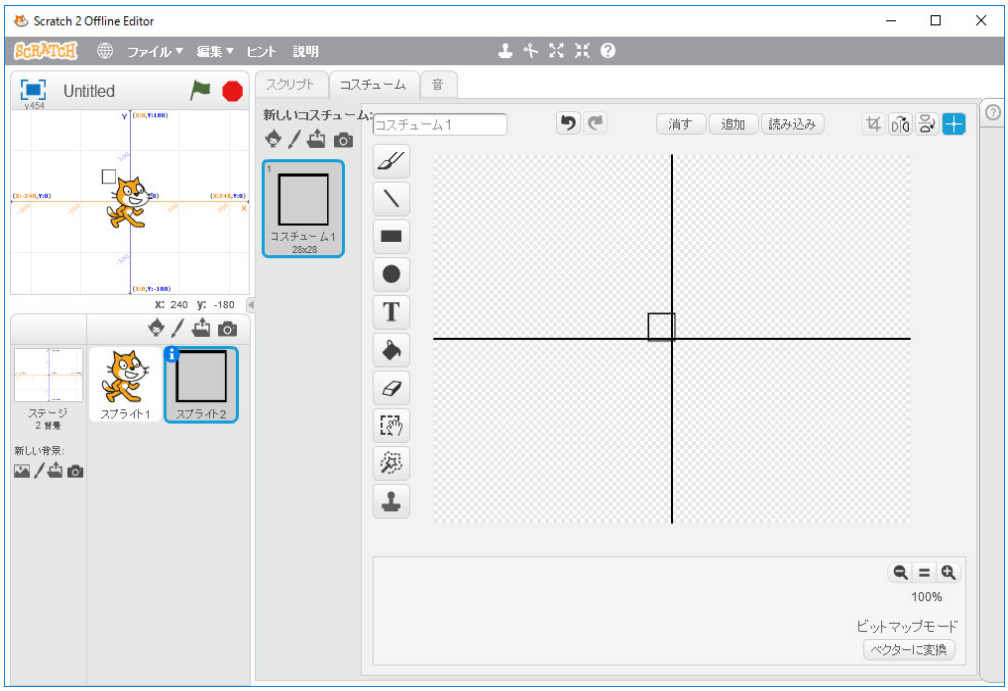
この中心に合わせて１４ドット×１４ドットの正方形のスプライトを作りたいわけです。正方形を描くには、左側の上から３番目の長方形のアイコンをクリックし、Shift キーを押しながらドラッグすれば、正方形を描いてくれます。後で微調整できますから、兎も角正方形を描いてみます。



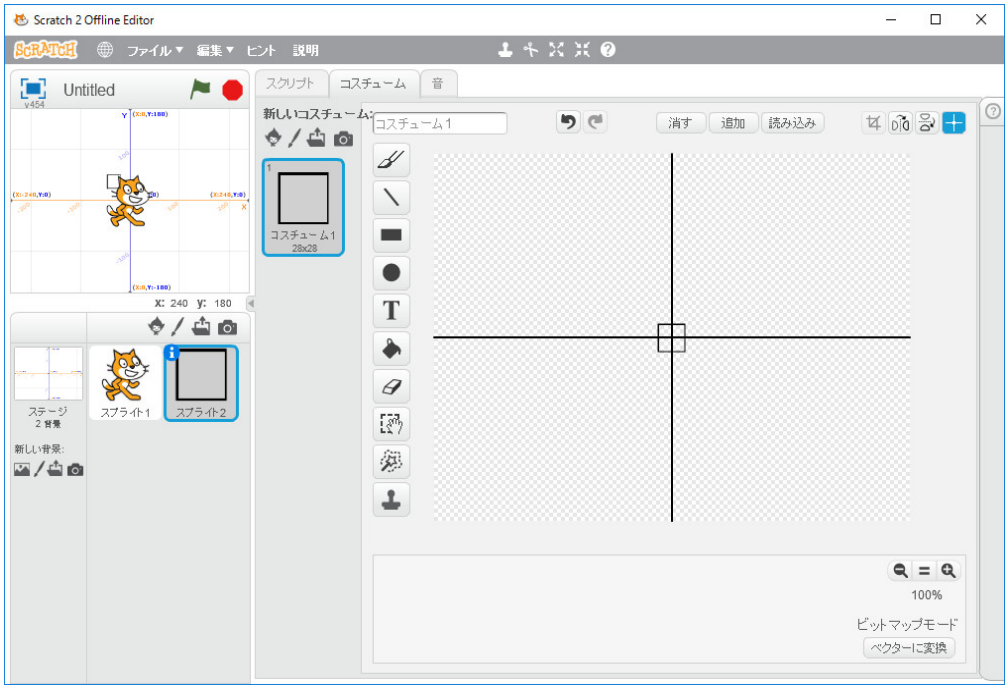
「コスチューム１」をクリックすると２８ドット×２８ドットの正方形のスプライトを描いたことが分かります。



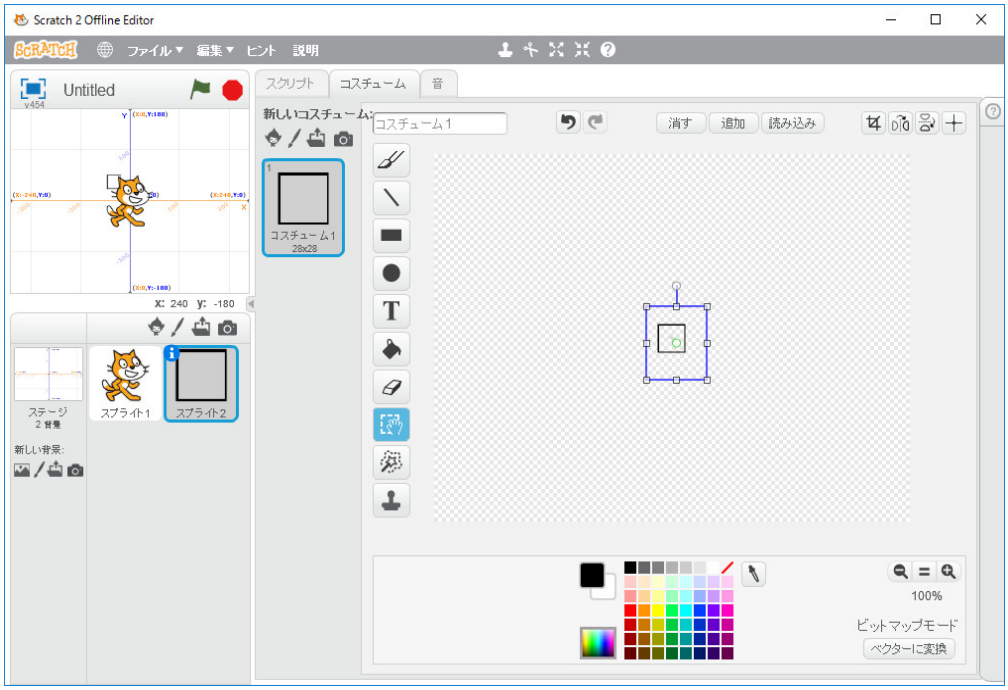
右上隅のプラスのアイコンをクリックすると



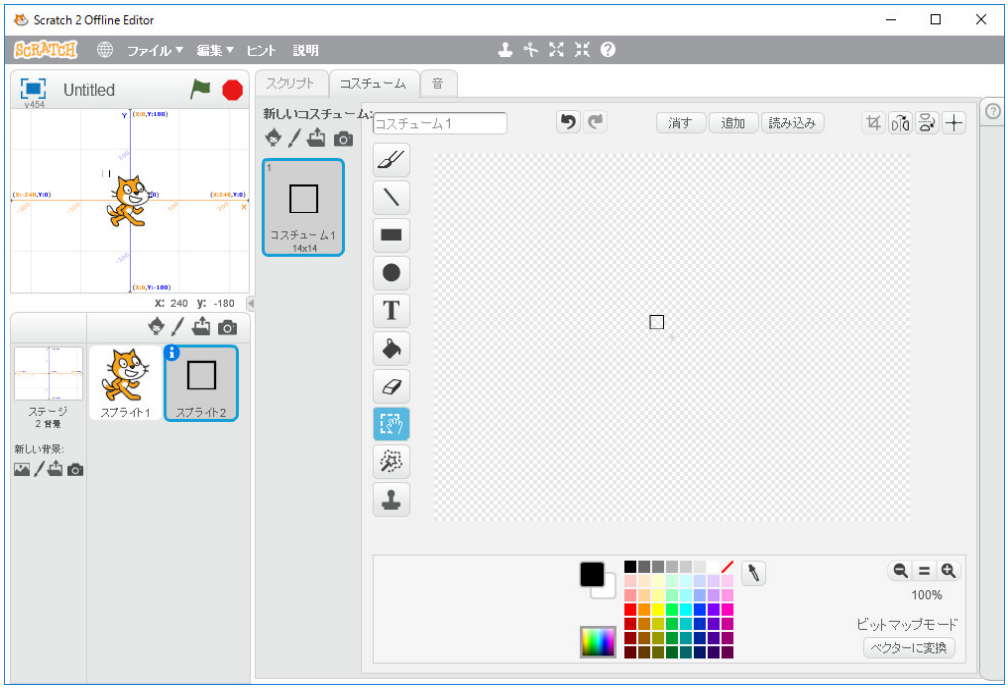
中心がずれています。マウスで線を動かし、中心だと思うところでクリックし、「コスチューム１」をクリックすると中心が移動します。



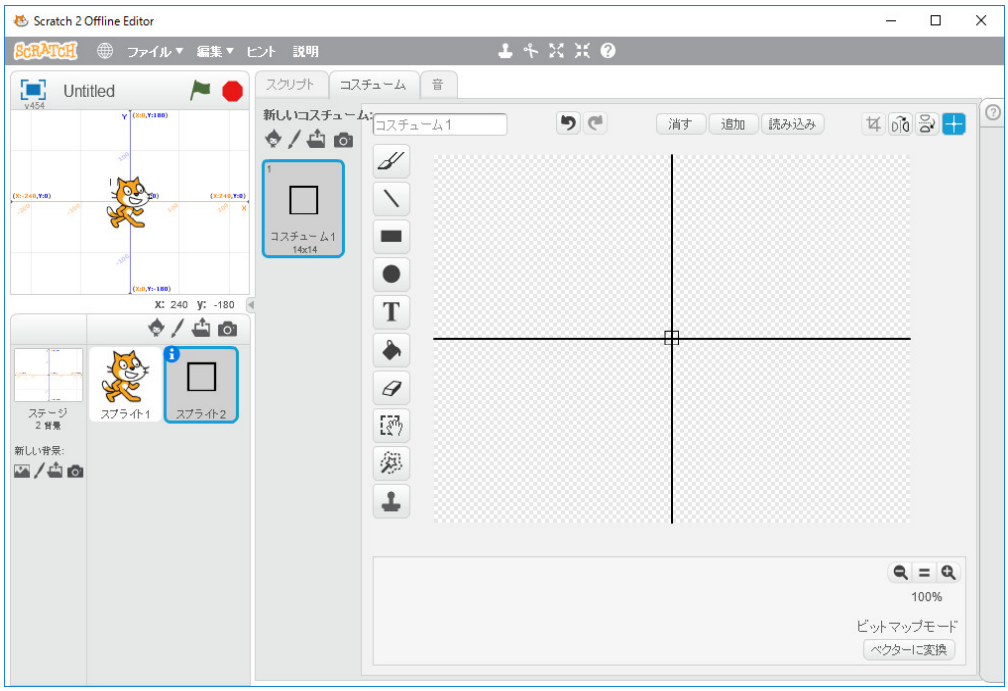
左側の上から８番目の「選択」のアイコンをクリックし、画面の正方形を含む範囲をマウスでドラッグすれば、



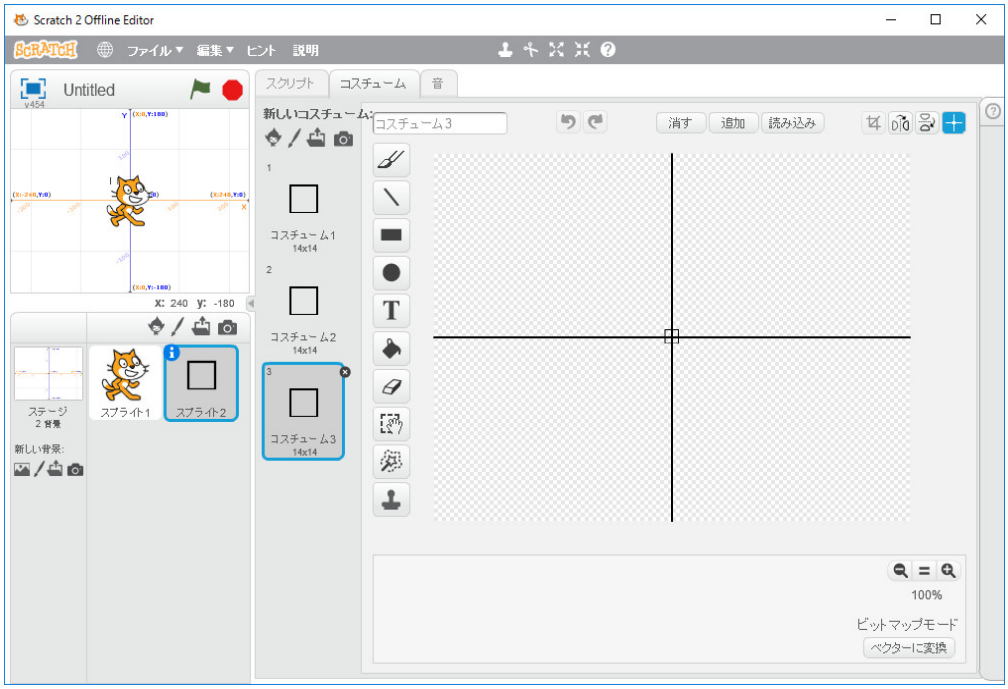
となり、大きさの微調整が出来ます。右下隅の正方形を操作して、１４ドット×１４ドットの正方形のスプライトに変更します。



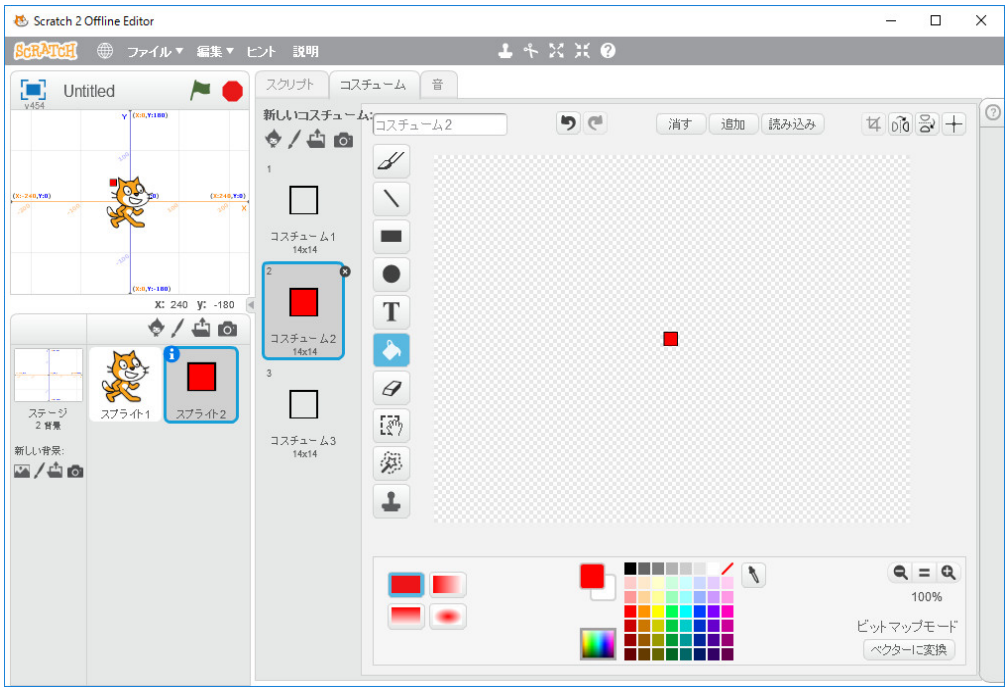
中心がずれています。マウスで線を動かし、中心だと思うところでクリックし、「コスチューム１」をクリックすると中心が移動します。



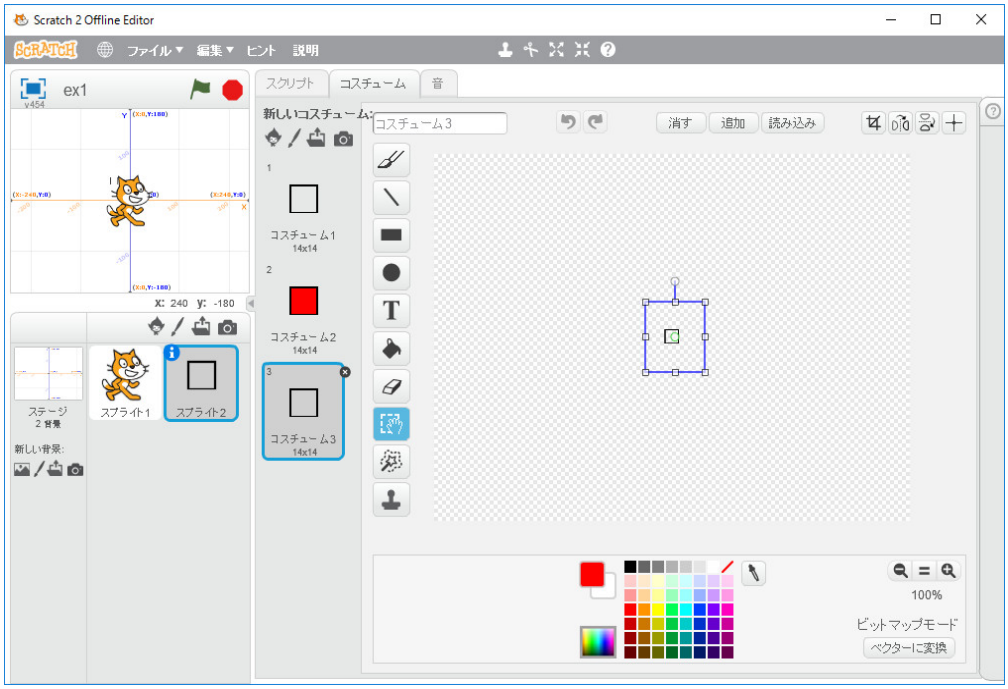
基本のスプライトが出来ました。せっかく苦労して作ったので、コスチュウムを複製しておいてから、以後の作業をします。「コスチューム１」を右クリックし、複製を選択すれば、コスチュウムの複製を作ってくれます。二つ複製を用心のため作っておきましょう。



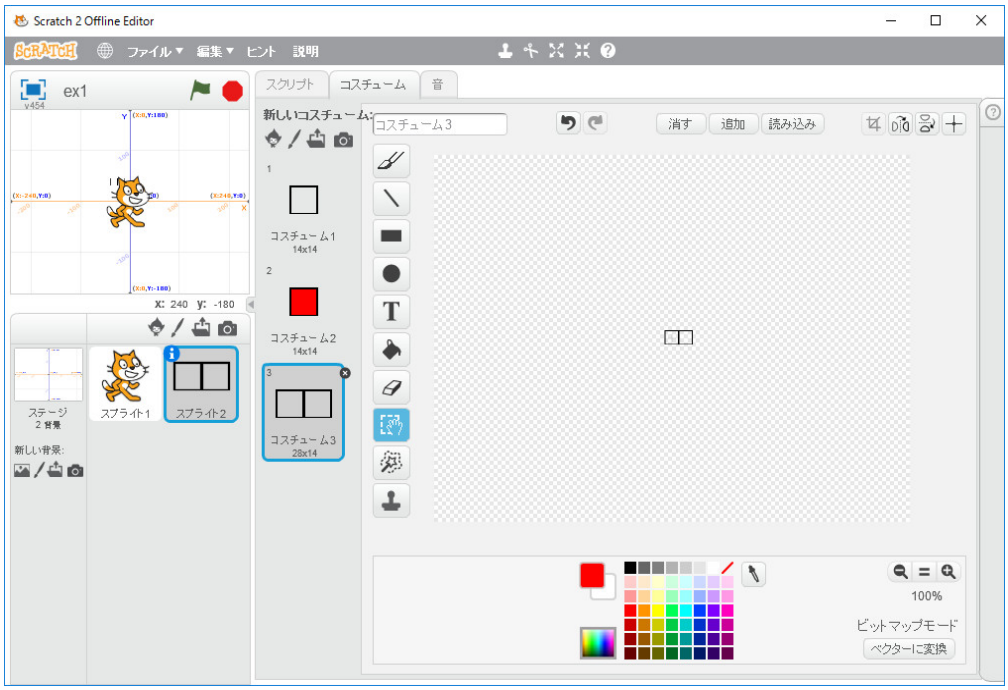
「バケツ」のアイコンをクリックし、赤で塗りつぶします。



テトリミノのスプライトを作るには、一番下の「選択して複製」をクリックし、正方形をマウスでドラッグして選択し、



横に移動すれば

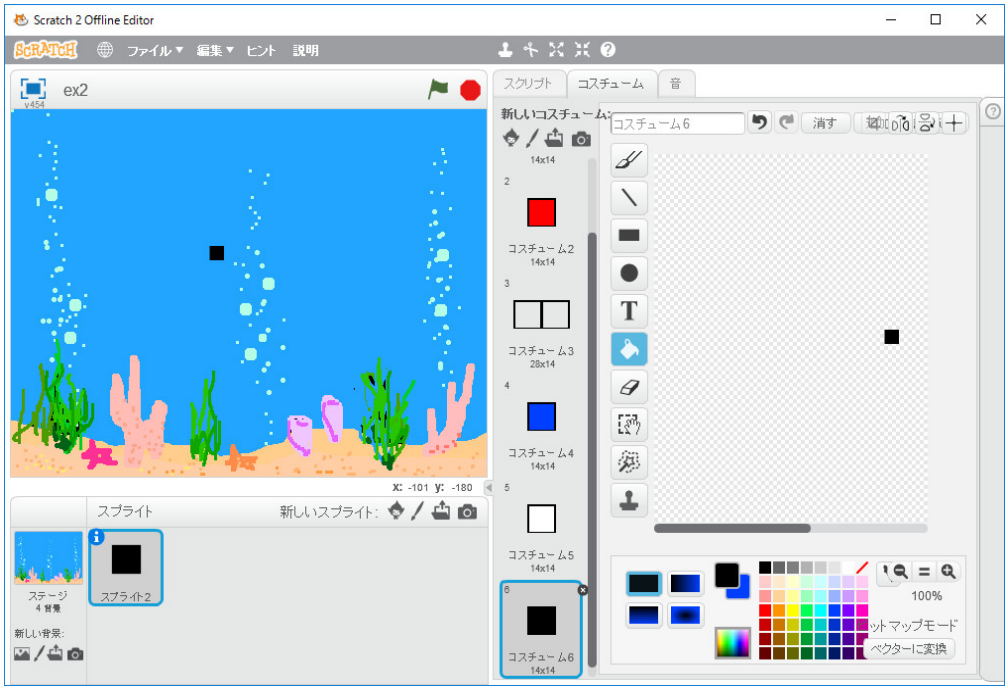


となります。このように複製してくっ付けていけば、テトリミノのスプライトを作ることが出来そうです。めんどくさいです。ぷよぷよのスプライトならこのような作り方でも良いかもわかりませんが、テトリスではスプライトは７種類のテトリミノ×４種類のコスチュームが必要で、絵を描くことが趣味である人たちは苦にしないと思いますが、私にはそんな根気はありません。正方形のスプライトをプログラムで４個組み合わせてテトリミノを描くことにしましょう。

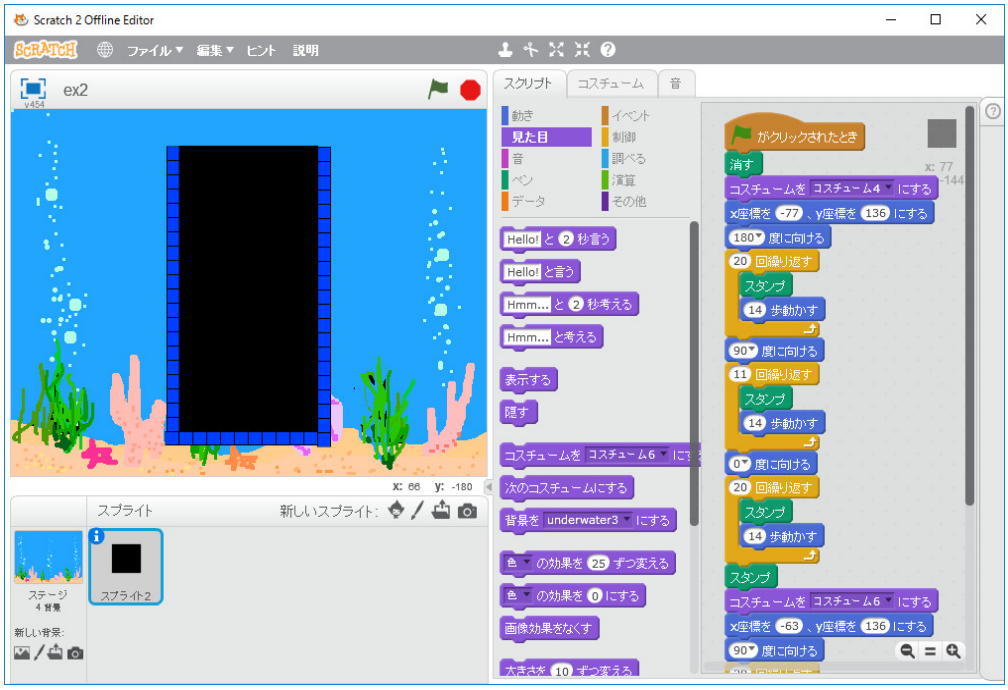
　次は背景をどのようにして作るかです。スプライトと同様、Scratch のお絵かきソフトで作るこ

とも出来ますし、別のグラフィックスソフトで絵を描いて、その画像を背景として取り込むことも出来ますが、お絵かきソフトで絵を描いた経験がないのでそれは断念します。Scratch には Scratch で描いた画面を保存して、背景とする方法が準備されているので、Scratch で背景を描き、それを使うことにします。

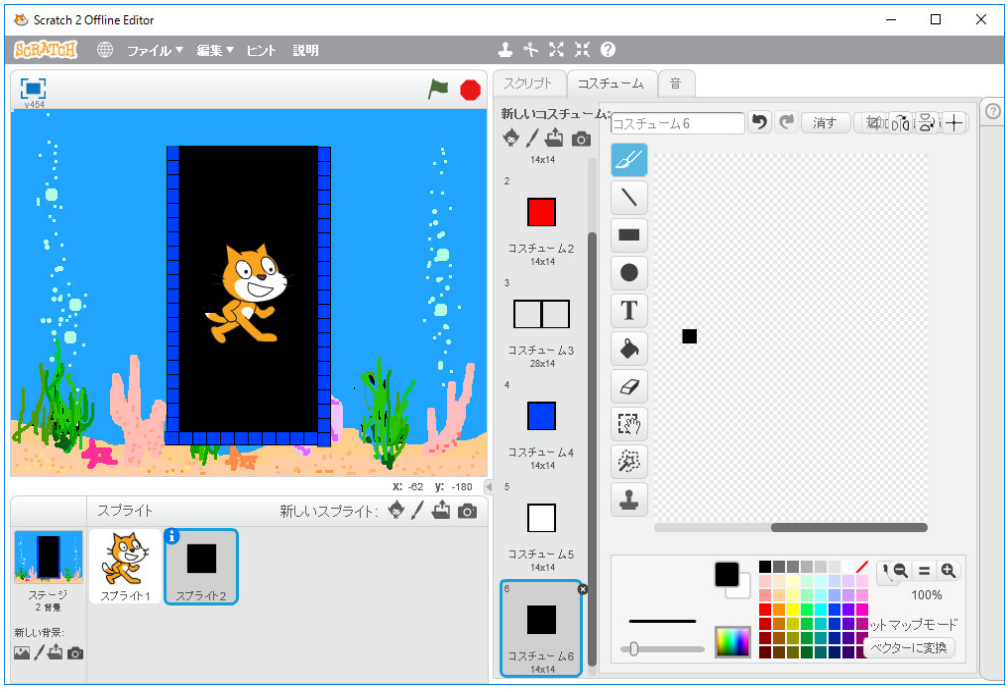
Scratch で準備されている適当な背景を選び、猫はいらないので、右クリックし削除します。正方形のコスチュームで、多分青（私は色弱で色の名前が良く分かりません）で塗りつぶされたものと白で塗りつぶされたものと黒で塗りつぶされたものも作っておきます。



中央に７種類のテトリミノを落とす場となる縦２０行、横１０列のフィールドを Scratch で描きます。



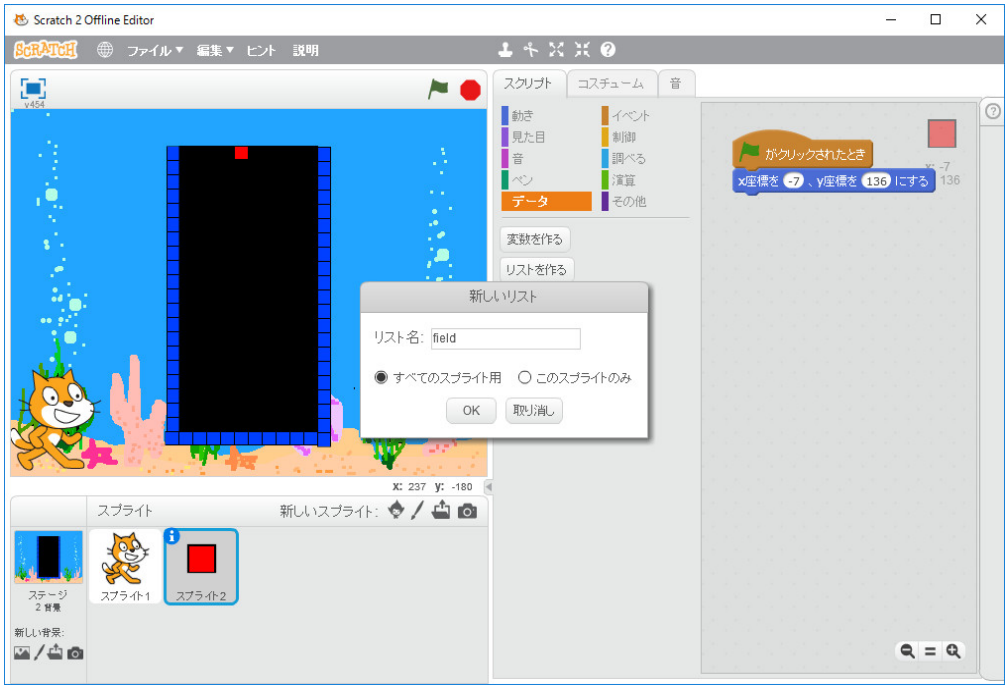
この画面を保存して、背景として使います。画面を右クリックすると「save picture of stage」と表示されるので、適当な名前を付けて保存します。更に、スプライトを右クリックして、「ローカルファイルに保存」を選択し、適当な名前を付けて保存します。新たに Scratch を立ち上げます。新しい背景の「ファイルから新しい背景をアップロード」のアイコンで、上で保存した背景の画像を読み込みます。次に、新しいのスプライトの「ファイルから新しいスプライトをアップロード」のアイコンで、上で保存したスプライトを読み込みます。



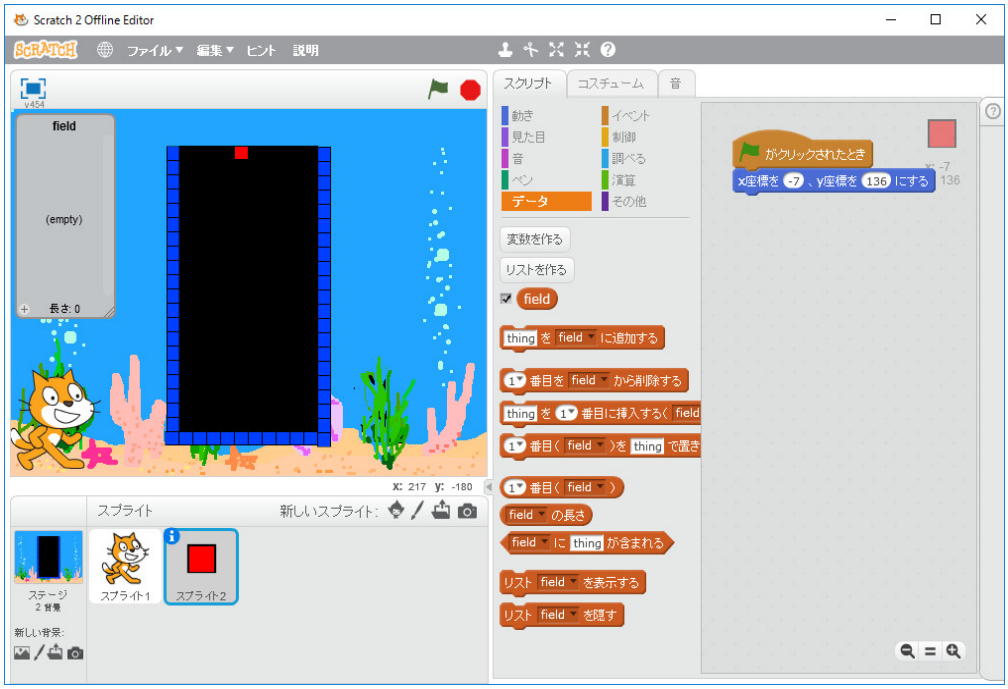
背景のためのスクリプトもロードされるので削除します。

平山尚著「プログラムはこうして作られる　プログラマの頭の中をのぞいてみよう」秀和システムを参考に、簡単なことからプログラミングしていきましょう。平山尚さんの sunaba のプログラムと Scratch のプログラムは、言語の文法が全然異なりますから、そのまま翻訳できませんのでプログラムを作り上げてから、整理して、説明するより、何を考えてプログラムを作っているかをその都度お見せした方が参考になると思います。

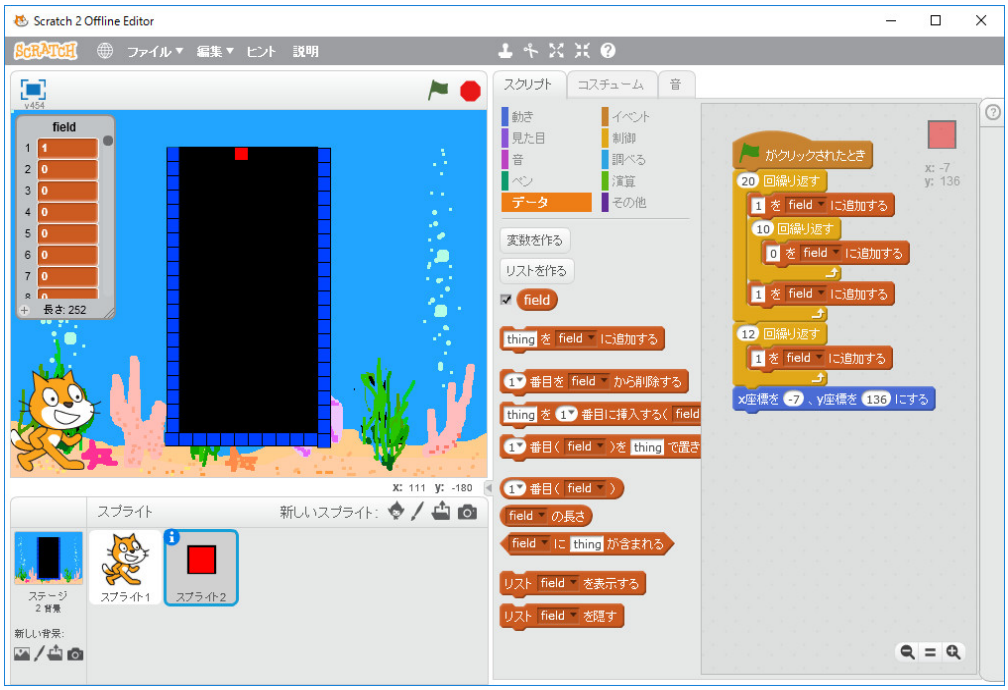
まず、「赤い四角のスプライトが落下して、白い四角になって積みあがっていくプログラム」を作ります。赤い四角のスプライトをｘ座標－７、ｙ座標１３６に置いて、落下させます。フィールドの壁と底及び積みあがった白い四角の位置を保持するために、Scratch の持っているリストを使います。横 12、縦 21 の矩形の状態を 0（空き）か 1（配置）の値で保持するため長さ 12 × 21 のリスト field を準備します。スクリプトの「データ」をクリックし、「リストを作る」をクリックします。



リスト名を field とし、「ＯＫ」をクリックします。



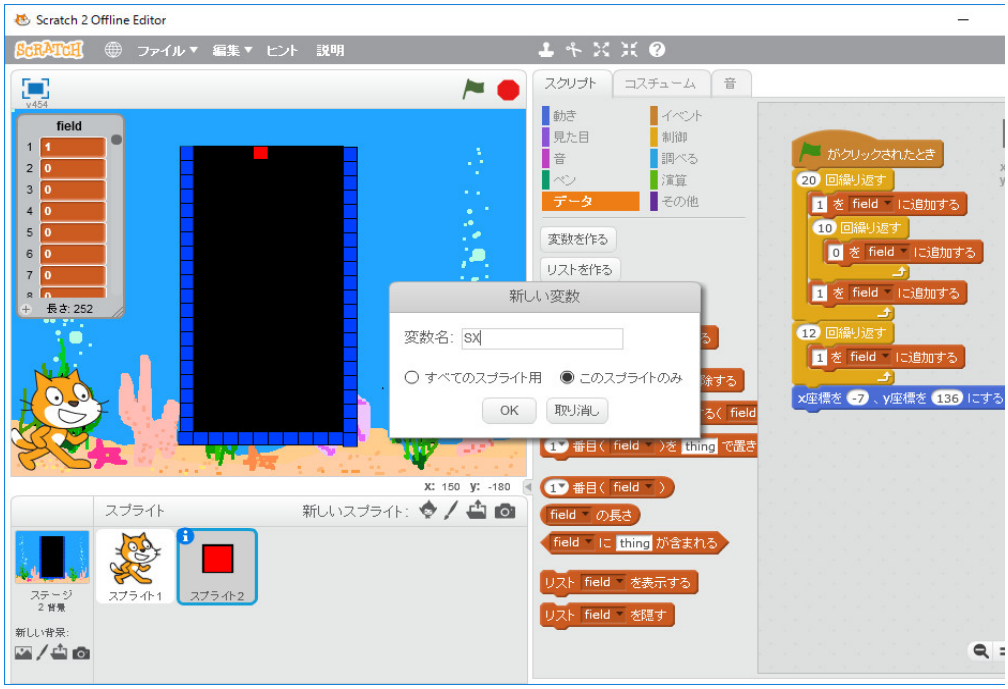
field をまず初期化します。



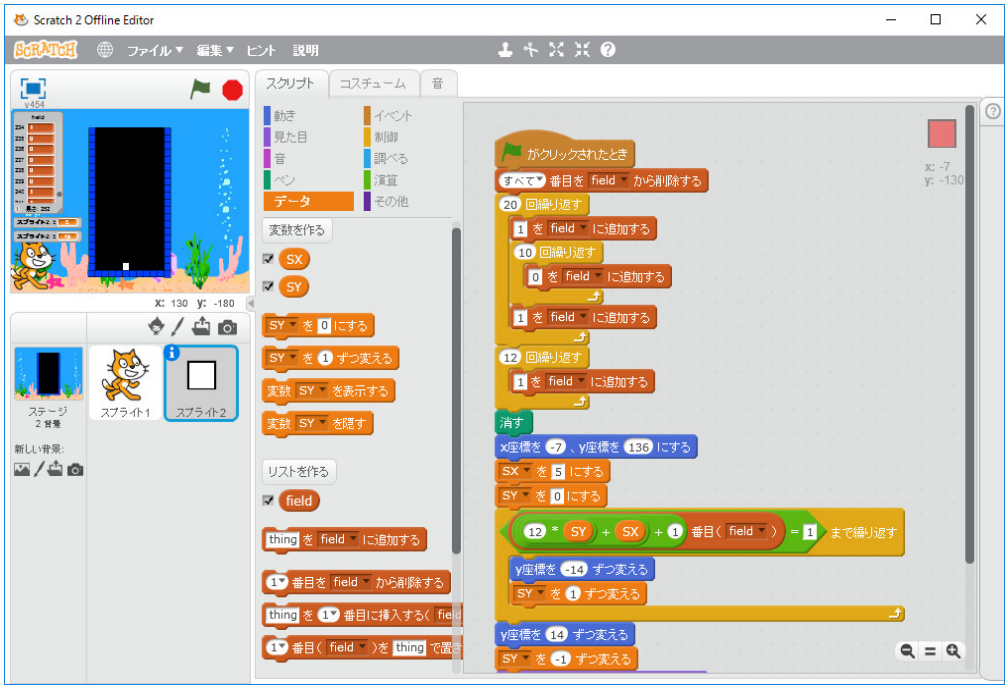
厄介なことに Scratch のリストは他のプログラミング言語と異なり 1 から始まっています。field の表示を消すには、「データ」フィールドの field の横のチェックを外せばいいです。しばらくこのままにしておきます。

スプライトを動かす操作はスプライトのスクリプトの「動き」の項目を見ればわかるように、Logo のような相対座標による方法と x 座標、y 座標の変更による絶対座標による方法があります。回転も後で考慮しないといけないので、絶対座標による方法を使うことにします。変数 SX,

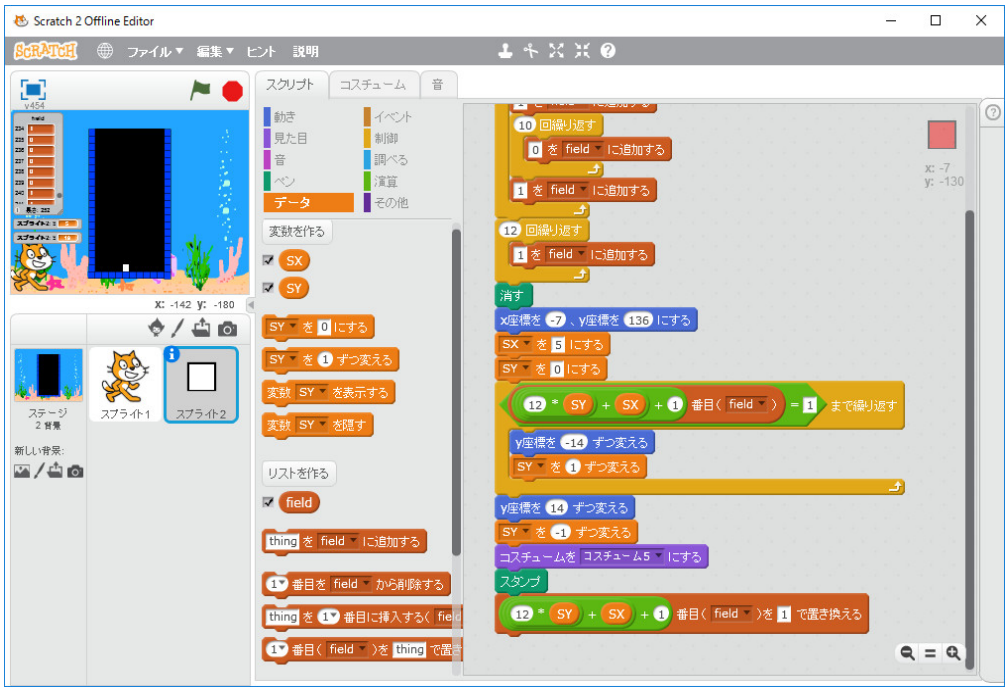
SY で field のどこにスプライトが位置しているか保持します。この座標はフィールドの左上隅を (SX,SY)=(0,0) とします。従って、最初赤い四角のスプライトをｘ座標－７、ｙ座標１３６に置いているので、フィールド座標は (SX,SY)=(5,0) です。スクリプトの「データ」をクリックし、「変数を作る」をクリックします。



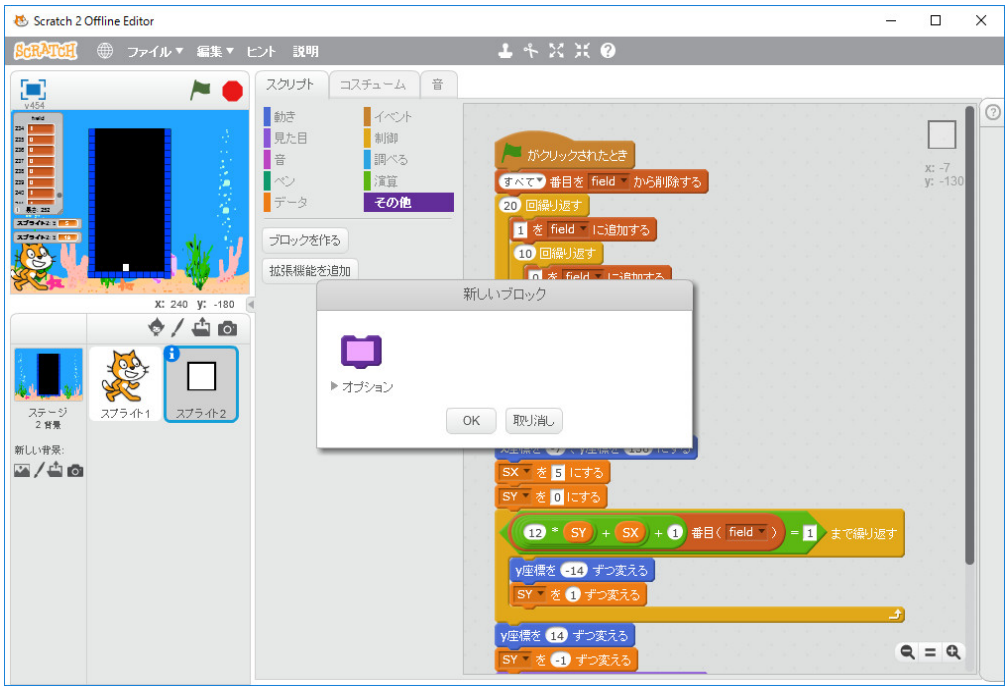
ローカル変数にするため「このスプライトのみ」をクリックして、「ＯＫ」を押します。同様に、変数 SY を作ります。「旗がクリックされたとき」の直後に、「すべて番目を field から削除する」を追加し



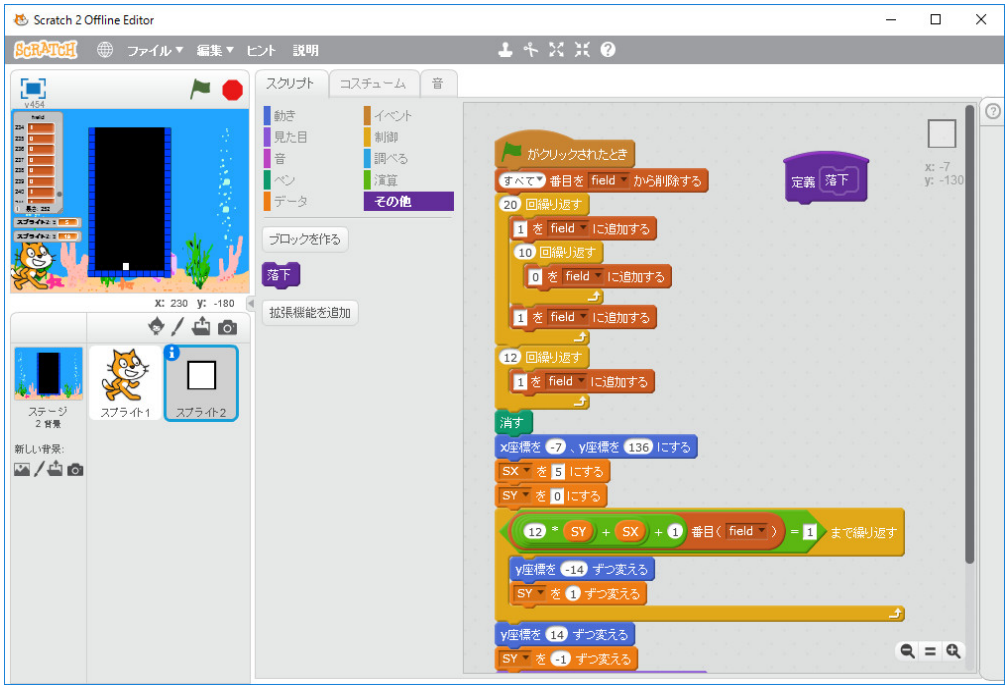
下図のようなプログラムを旗をクリックして実行すると



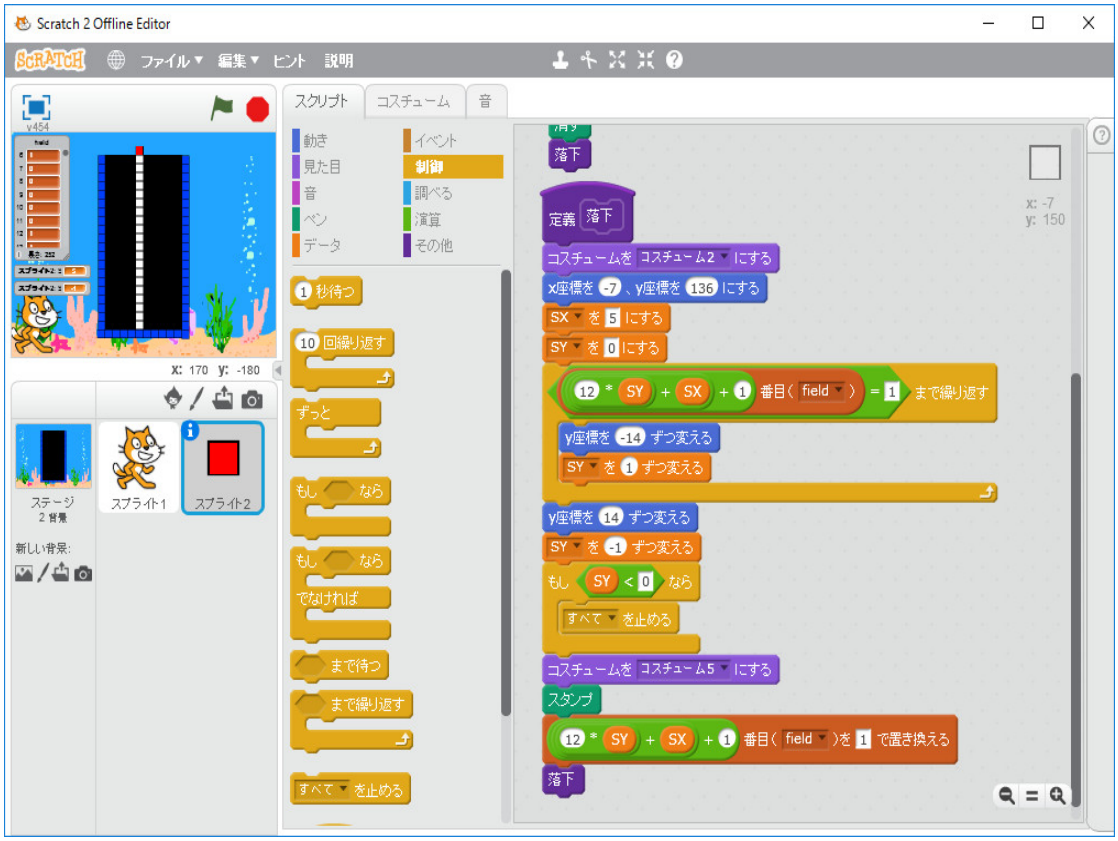
となります。スプライトを落とす部分を「ブロック」（通常のプログラミング言語のプロシージャ）にします。スクリプトの「その他」をクリックし、「ブロックを作る」をクリックします。



名前「落下」をアイコンの枠の中に入力し、「ＯＫ」をクリックします。



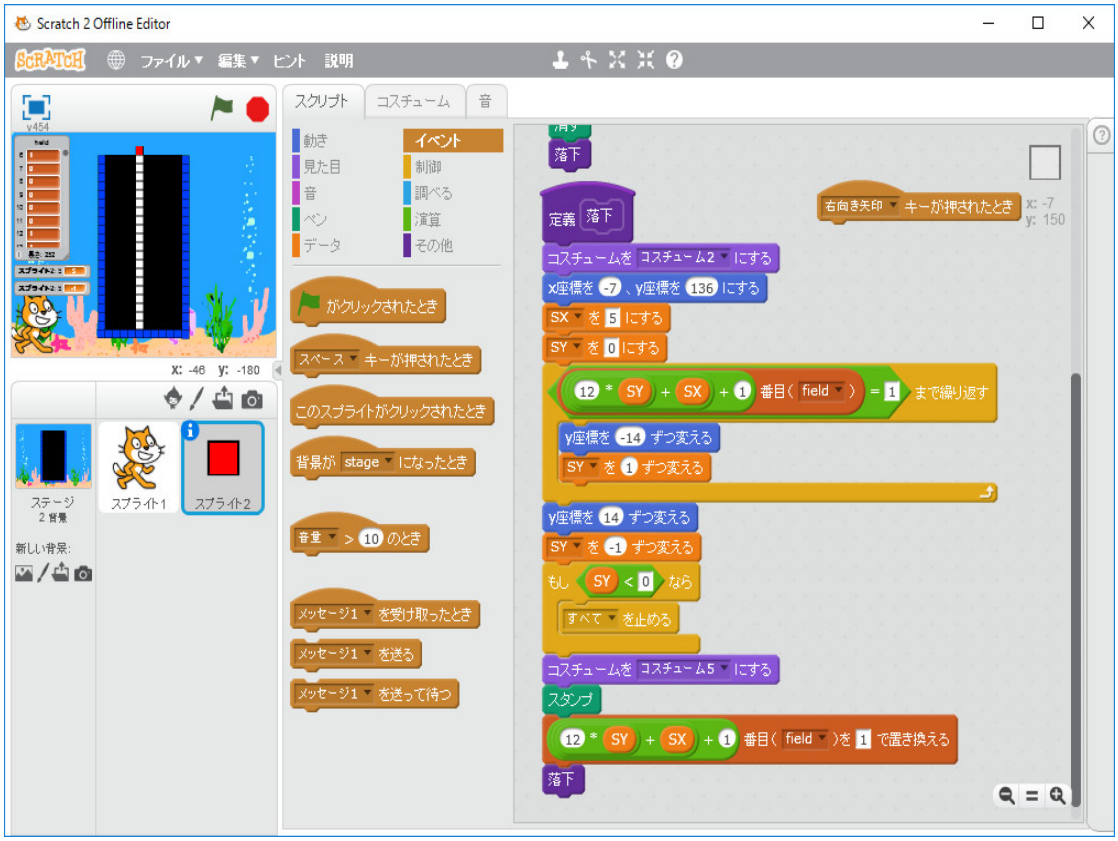
「消す」以下を図のように修正して「落下」の定義以下に回し、「消す」の下に「落下」を追加し、実行すると図のようになります。



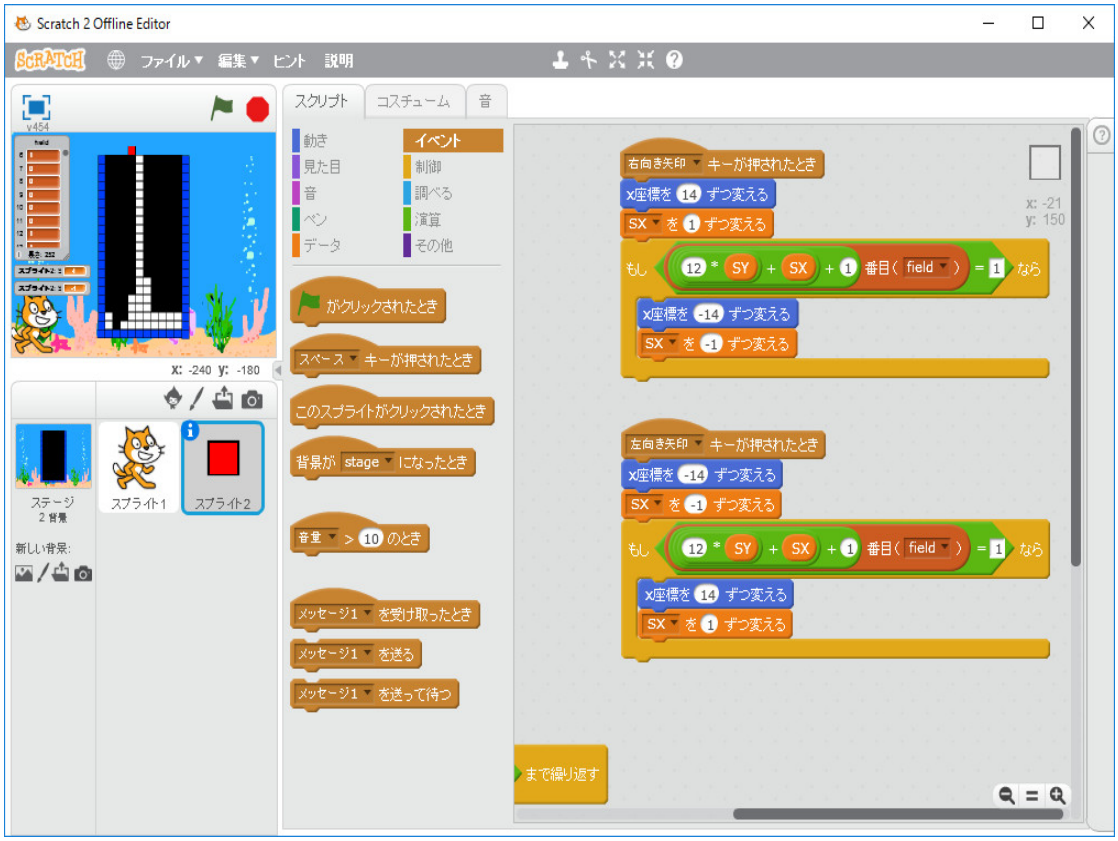
Scratch の「ブロック」（通常のプログラミング言語のプロシージャ）はこのように再帰呼び出し

が可能です。

　次は矢印キーで動かすことが出来るようにします。「イベント」の「スペースキーが押されたとき」を「右向き矢印キーが押されたとき」に変更して使います。



次の図のようにプログラミングし、「イベント」の「左向き矢印キーが押されたとき」のプログラムも同様に作ります。



キーによる操作ができるようになりました。落下のスピードが速すぎれば、「制御」の「１秒待つ」を使って、秒数を指定し、遅くすればいいです。

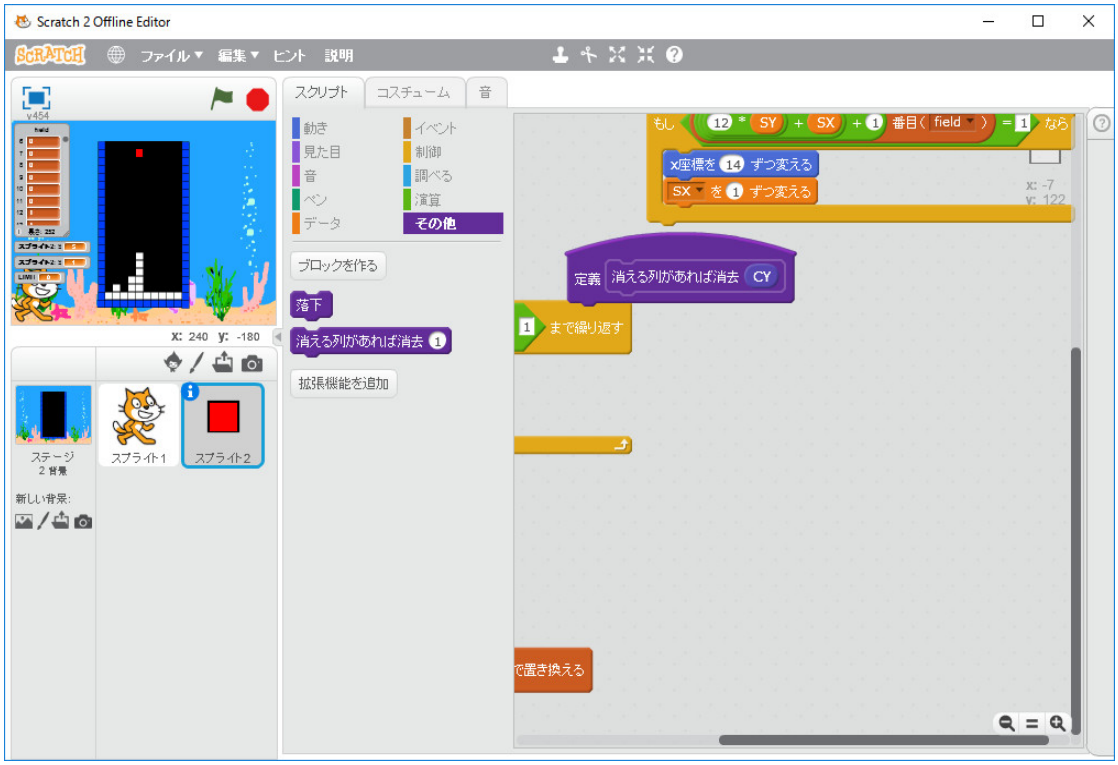
　次は、横に一列並べば、その列を消して、上の四角を下に落とすプログラミングをします。テトリスでは最大４段消えるので、それに対応できるようにする。「落下」のブロックの最後の「落下」の前に「消える列があれば消去」のブロックを挿入する。

「オプション」をクリックし、「数値の引数を追加」の右のアイコンをクリックする。

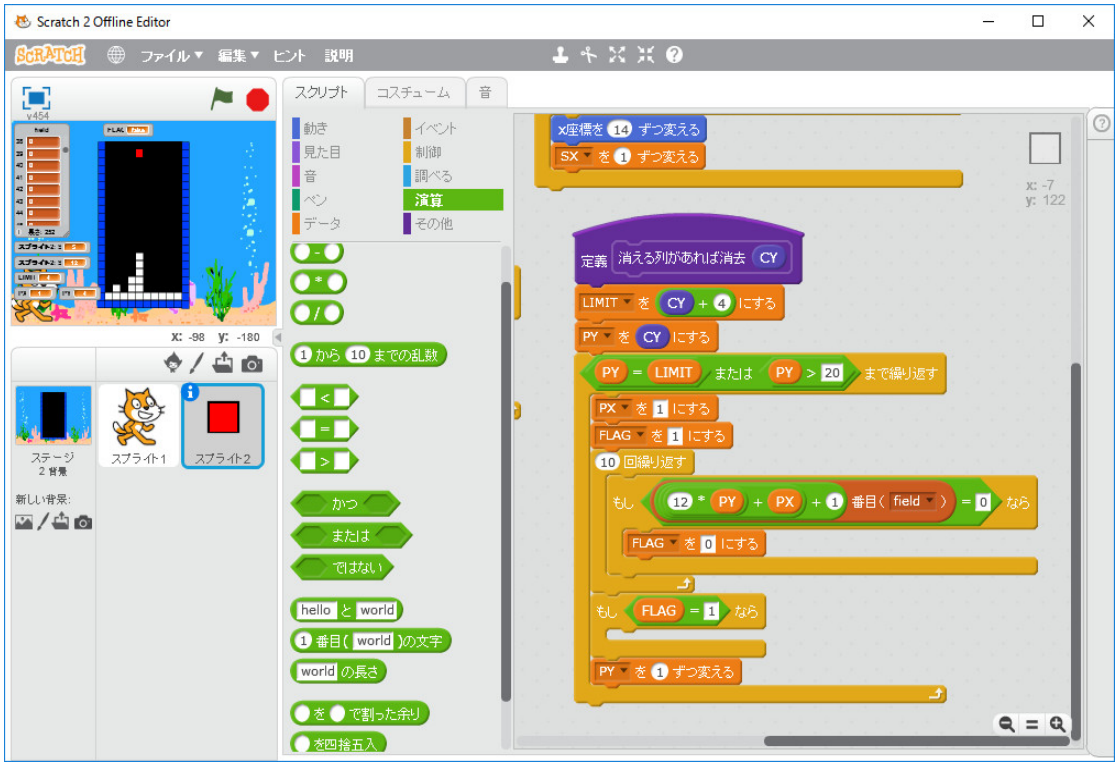
number1 を CY に変える。

「ＯＫ」をクリックする。

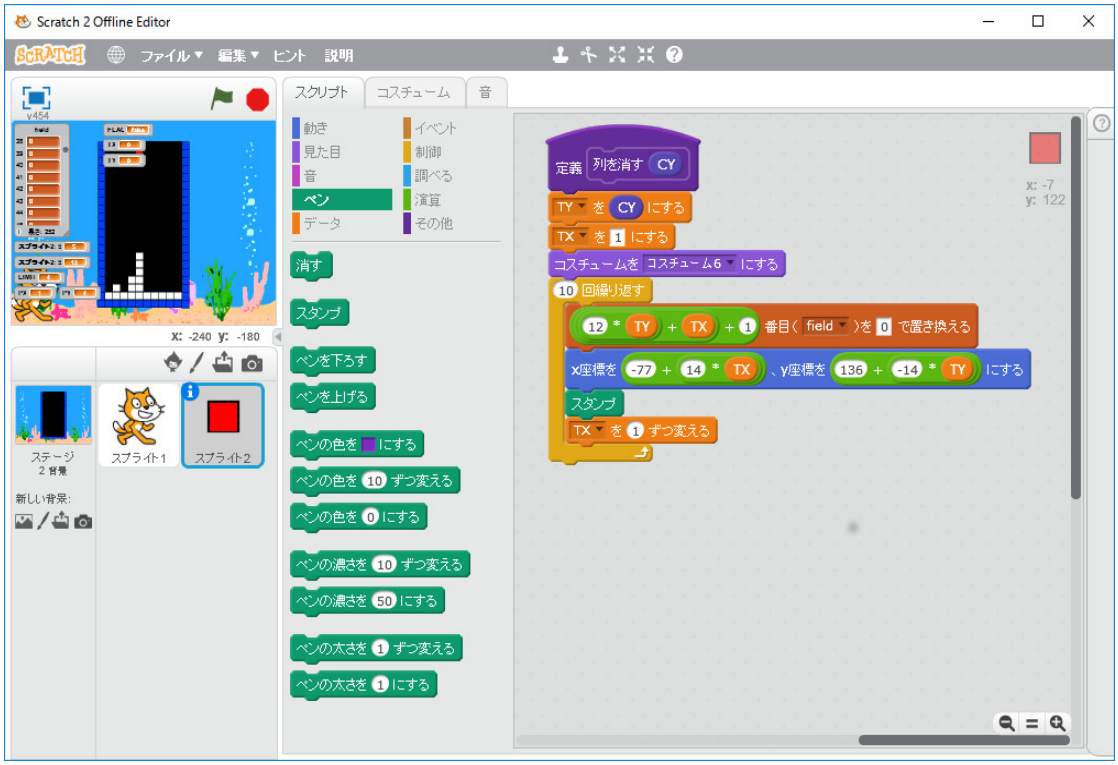
これで１つの引数を持つブロック「消える列があれば消去」を定義できました。



消える可能性のある一番上の y 座標を CY にセットします。ブロックのローカル変数の概念がないので、スプライトの変数 LIMIT と PX と PY と FLAG を定義します。ブロック「消える列があれば消去」をまだ途中ですが

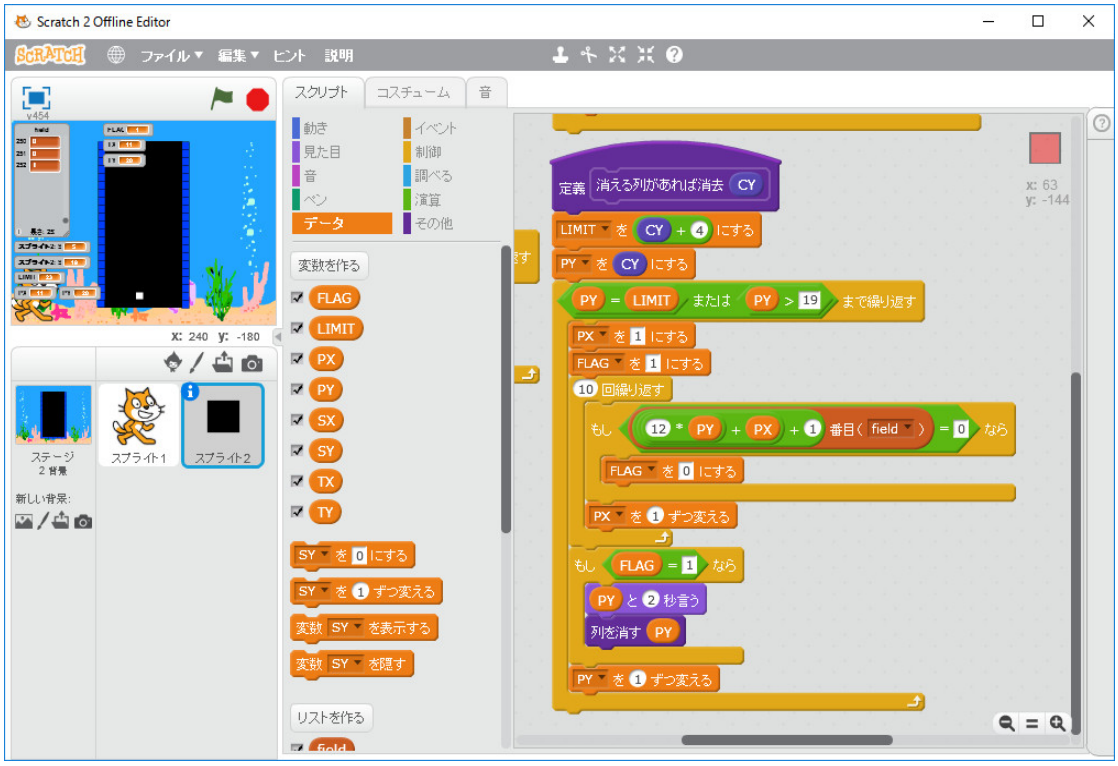


と定義します。制御構造が貧弱なので工夫する必要があります。「1=1」は true、「0 > 1」は false となりますが、これを保存する変数がブロックの引数しかないので、true の代用に 1 を、 false の代用を 0 とします。ただし、計算式の中では数値として true は 1 、 false は 0 として通用します。「もし FLAG = 1 なら」の空いている所にブロック「列を消す」と「上を落とす」を挿入します。まず、ブロック「列を消す」を作ります。引数は CY とします。制御変数 TX と TY を作ります。「もし FLAG = 1 なら」の空いている所に「列を消す（PY)」挿入し、ブロック「列を消す」を次のように定義します。



「落下」のブロックの最後の「落下」の前に「消える列があれば消去 (SY)」のブロックを挿入して、実行してみます。間違っています。

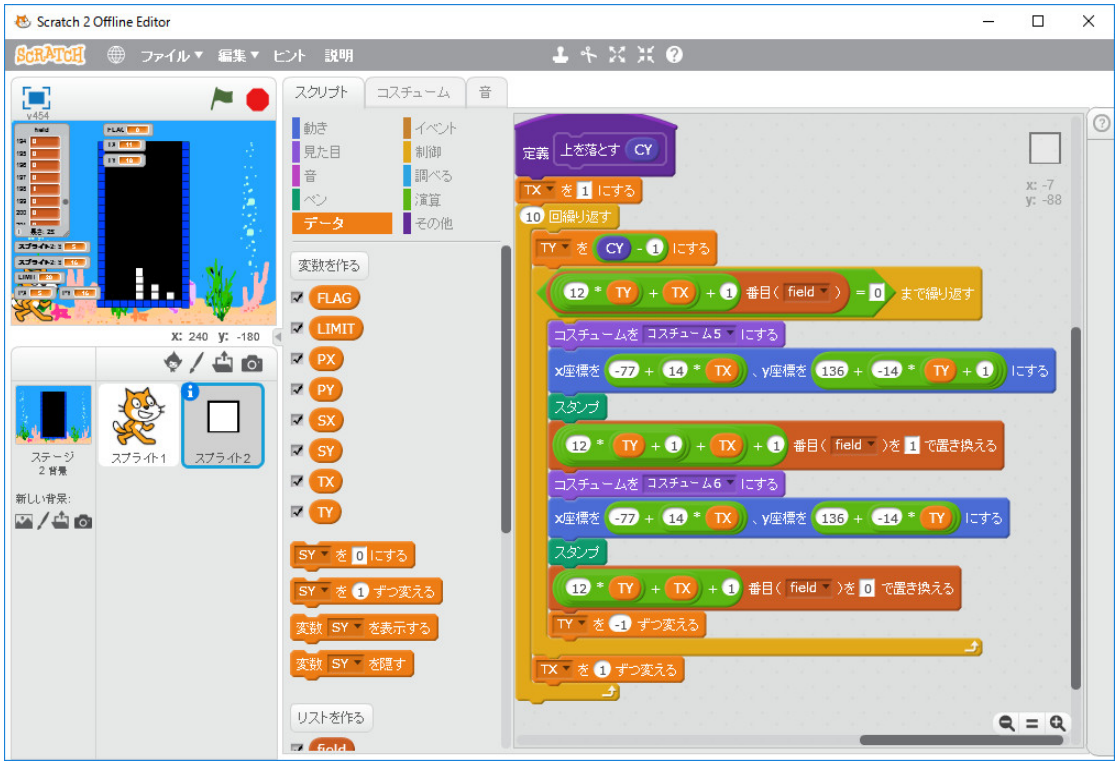
底が消えました。



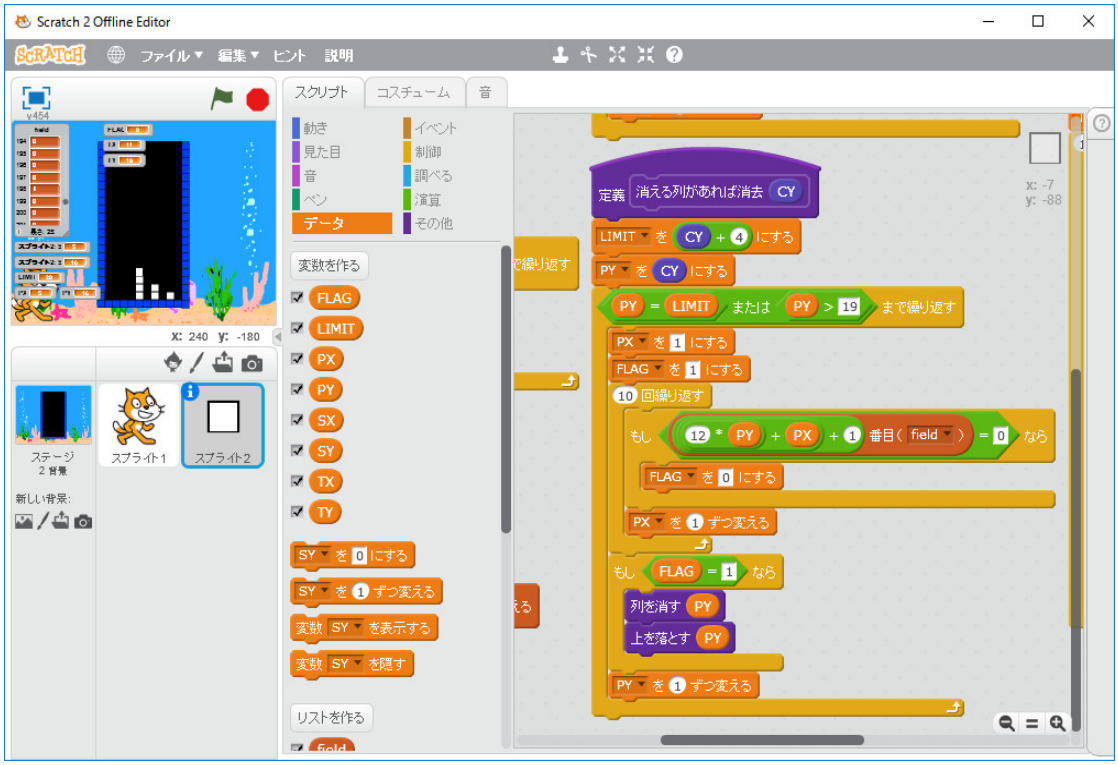
列を消す直前に「PY と 2 秒言う」という命令を入れて実行してみると PY=20 であることが分かります。これは底の列です。上の図で修正しているように「PY=LIMIT または $PY > 20$ まで繰

り返す」となっていたのを「PY=LIMIT または $PY > 19$ まで繰り返す」に修正します。実行してみます。

上手くいっているみたいです。次にブロック「上を落とす」を作り、プログラミングします。消した列の y 座標を引数として渡します。プログラムは



です。ブロック「消える列があれば消去」の「列を消す」の次に「上を落とす（PY）」を追加します。

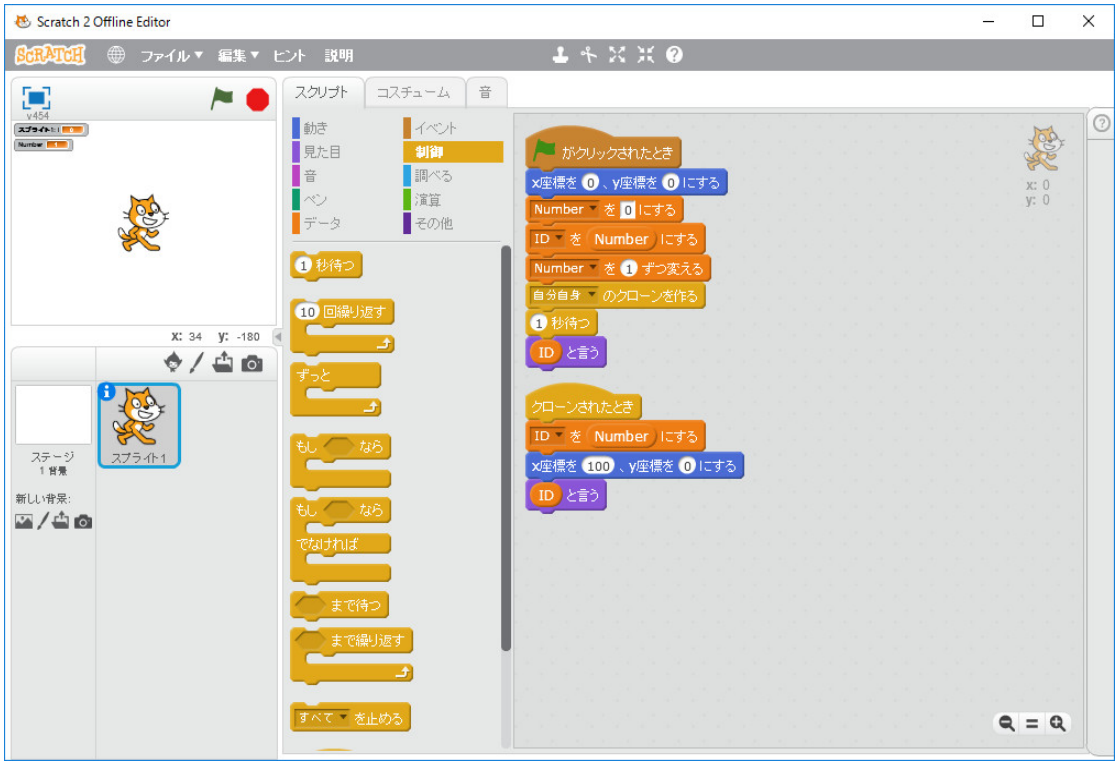


実行すると上手くいっているみたいです。唯、ブロック「消える列があれば消去」をしている間に矢印キーを押すと落下している四角が移動するので、この処理の間は四角のスプライトを隠しておく方がいい。ブロック「消える列があれば消去」を「隠す」と「表示する」で挟む。

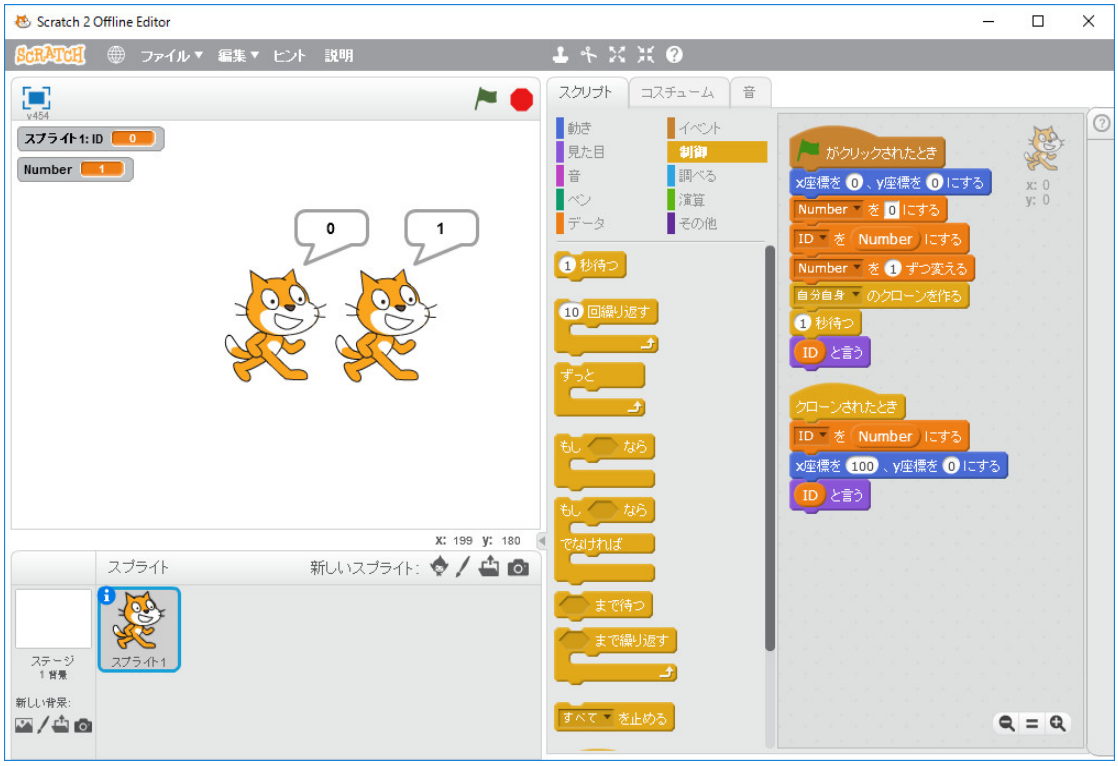
用心のため、プログラムを名前を付けて保存しておく。

次は四角４個の７種類のテトリミノを使う場合の考える。考え方は二通りある。四角の別のスプライトを使う方法と今使っているスプライトのクローンを作って使う方法が考えられる。

まずは楽そうなスプライトのクローンを作って使う方法を試してみる。SX と SY を「このスプライトのみ」とローカルな変数にしたのですがこれが何を意味するか正確に理解してないので、ちょっと実験をしてみます。新しく Scratch を立ち上げ、スプライト 1 の変数 Number を「すべてのスプライト用」として、変数 ID を「このスプライトのみ」として作ります。プログラムを次のように作ります。

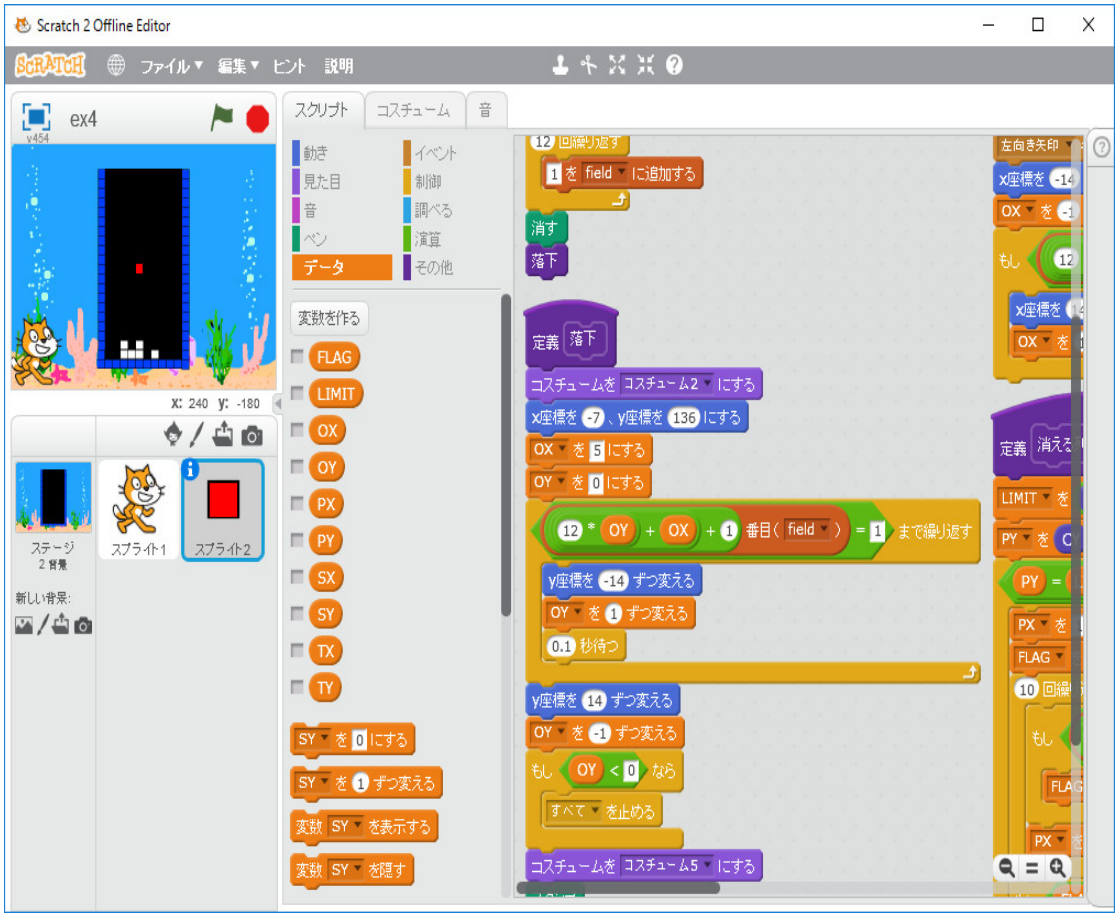


実行すると



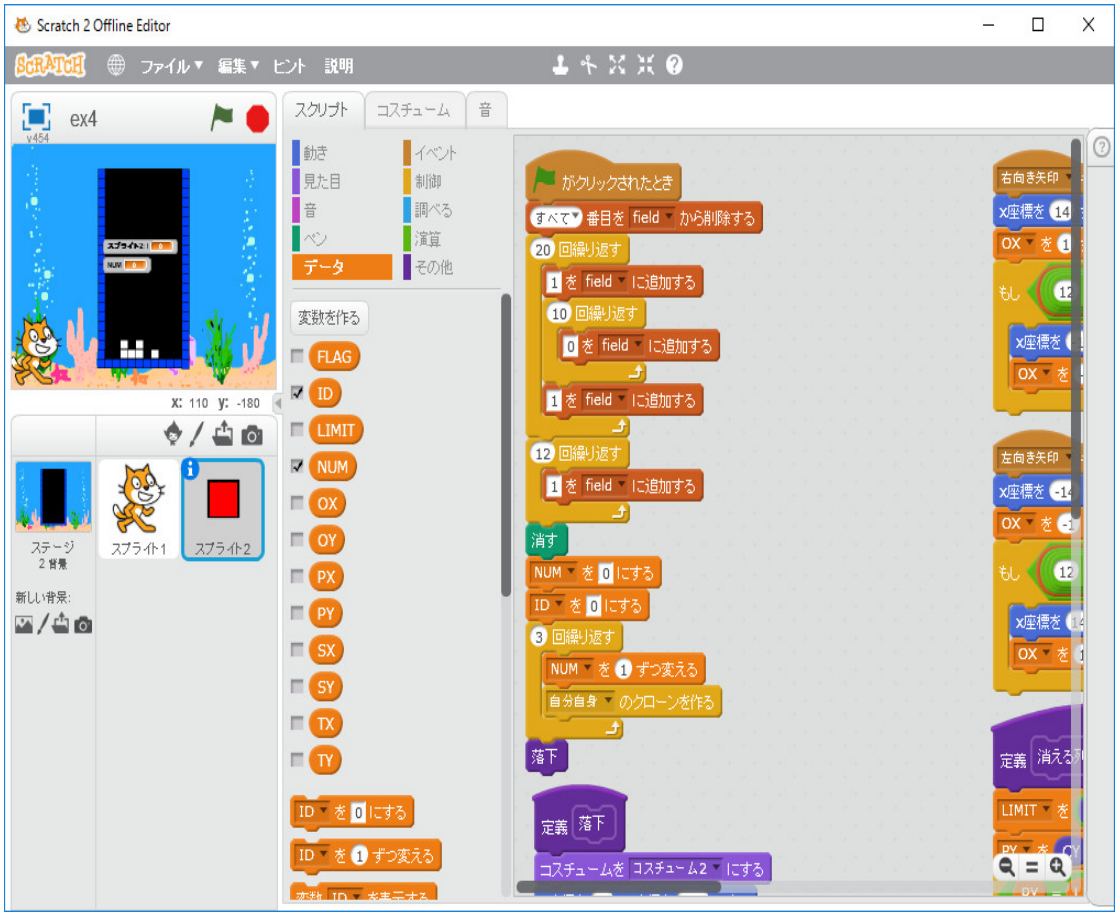
です。元のスプライトの変数 ID はクローで値を変えた後でも変化していません。「このスプライトのみ」で作った変数はクローンごと異なる値を持っています。従って、オリジナルな四角のスプ

ライトの SX, SY の値は、四角のスプライトのクローンでは知ることが出来ません。オリジナルな四角のフィールド座標の位置が分かるよう新たに「すべてのスプライト用」の変数として OX と OY を作ります。ここでまた選択肢が２つあります。SX, SY の値が変化したところで、その値を OX, OY にコピーする方法（この場合、８か所修正）とすべての SX, SY を OX, OY に修正する（この場合、１７か所修正）です。前者の場合、修正箇所は少ないが、プログラムが長くなり、従って、実行速度が遅くなる。後者の場合、修正箇所は多いが、プログラムの長さは同じで、実行速度も変わらない。ここでは後者の方法を取ります。プログラムを修正します。全部は表示しませんが

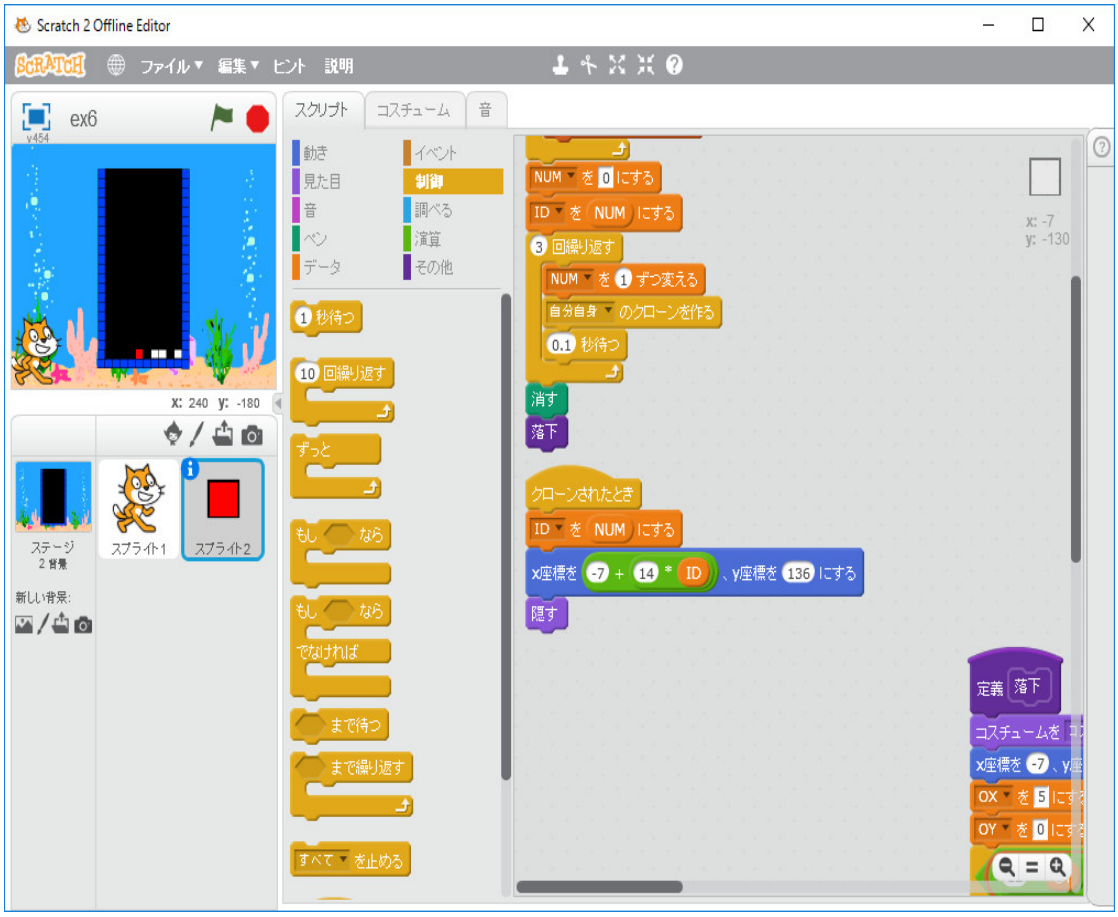


のようになります。

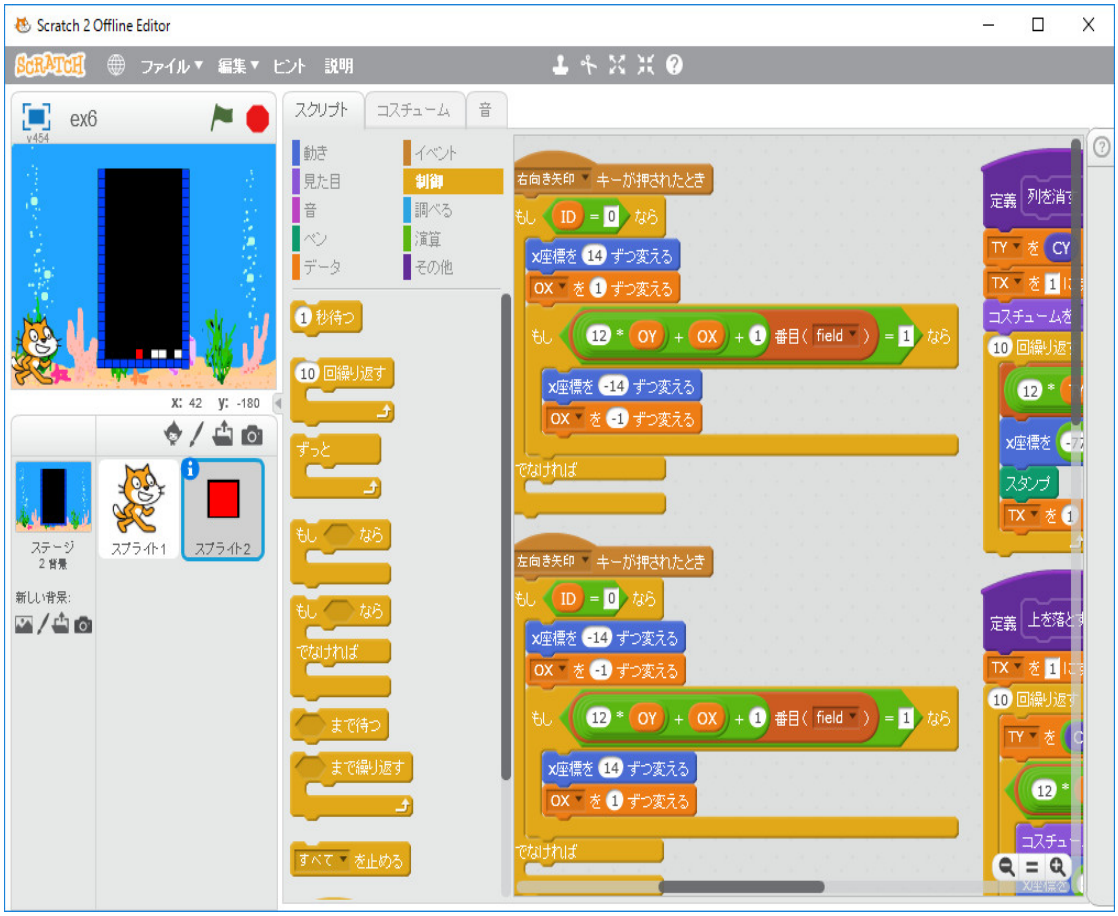
３個のクローンを作り、オリジナルな四角と一緒に行動させるようにする。クローンを区別するために、変数 ID を「このスプライトのみ」とローカルな変数として作る。ID を割り振るために、「すべてのスプライト用」の変数として NUM を作る。３個のクローンを ID=1,2,3 として作るプログラムを「旗がクリックされたとき」の「消す」と「落下」の間に挿入する。



「クローンされたとき」の処理を次に作ります。しかし、クローンが生成された時には、ID をセットし、位置は仮にオリジナルの横に並べ、「隠す」を実行するだけにします。更に、混乱しないように、「自分自身のクローンをつくる」の後に「０．１秒待つ」を入れておきます。



実行して、上手く動くか確かめてみます。上手く動きません！左右の矢印キーを押した時、端まで行きません。これは「右向き矢印キーが押されたとき」と「左向き矢印キーが押されたとき」の処理をクローンもやっています。オリジナルな四角の処理が今書かれているプログラムでクローンのためのプログラムはそれぞれ書かなければいけなかったわけです。応急処置として、「右向き矢印キーが押されたとき」と「左向き矢印キーが押されたとき」の処理を次のように ID=0 （オリジナルな四角だけが実行するように）の時だけ実行するように修正します。



オリジナルの四角が表示された時、クローンも表示しなければなりません。そのためには、7種類のテトリミノのどれであるかと自分自身の ID に基づいて、フィールドのどの位置に配置すべきかを SX, SY にセットし、表示する。そのため、現在 7 種類のテトリミノのどれであるかを示す「すべてのスプライト用」の変数 KIND が必要です。更に、回転した場合の各クローンの相対位置の情報も必要です。平山尚著「プログラムはこうして作られる　プログラマの頭の中をのぞいてみよう」秀和システムでは、この情報に関し複雑な議論をしていますが、計算で相対位置を求めるより、表で与えて参照するほうが一般的言語（例えば、C++）では簡単だと私は思います。平山尚さんは「**そんな面倒くさいことをするわけがない**」、「**プログラマたるもの、面倒くさいと思うようなことを工夫なしにやってはいけない**」と書いてあります。しかし、我々凡人のアマチュアプログラマはとにかくプログラムを素早くでっちあげることだけを考えます。「すべてのスプライト用」のリストとして table を作る。tabele の内容は、y 座標、x 座標のペアを順に並べて

```
0 -1 0 1 0 2
-1 0 1 0 2 0
0 1 0 -1 0 -2
1 0 -1 0 -2 0

-1 -1 0 -1 0 1
-1 1 -1 0 1 0
1 1 0 1 0 -1
```

```
1 -1 1 0 -1 0

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

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

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

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

0 1 1 -1 1 0
1 0 -1 -1 0 -1
0 -1 -1 1 -1 0
-1 0 1 1 0 1
```

です。Scratch にはこのデータを一括で table にセットする命令がないので、１つ１つ、１６８個セットしていかなければなりません。C++ のようにデータ構造が豊富で、一括で table にセットする命令があれば、上の表を作ると同じ手間で、table にセット出来ます。**しかし table にセットする命令の１６８個のブロックを並べるのは、さすがにやっぱり大変です。せっかく平山尚さんが各ブロックの１行目だけで、後は計算で求める方法を説明してくれているので、それを使います。**

まず、１８０度の回転の法則を導いています。

法則その１：１８０度回すには、縦も横もプラスマイナスを入れ替えればよい！

この法則により、０度の回転と９０度の回転のデータだけあれば良いので表は半分で良くなります。

次に、９０度回した時の法則を導いています。平山尚さんの説明は初等幾何学的で分かり難いですが、我々は複素数を知っているので、これは複素数で考えると分かりやすいです。x 座標=x、y 座標=y の点は $x + \sqrt{-1}y$ で表されます。９０度の回転は $\sqrt{-1}$ を掛ければ良いです。 従って、

$$(x + \sqrt{-1}y) \times \sqrt{-1} = -y + \sqrt{-1}x$$

です。また、これは高等学校で習ったベクトルを使っても証明することが出来ます。ベクトルの回転は

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

で与えられるので、$\theta = \pi/2$ を代入して

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -y \\ x \end{pmatrix}$$

で、同じ結果を得ます。折角学んだ数学です。このようなときに使います。

従って

法則その２：９０度回すには、縦横を入れ替えてから、横のプラスマイナスを入れ替えればよい！

あるいは

法則その２：９０度回すには、縦のプラスマイナスを入れ替え手から、縦横を入れ替えればよい！

です。

この法則により、０度の回転のデータだけあれば良いので表は更に半分で良くなります。従って、

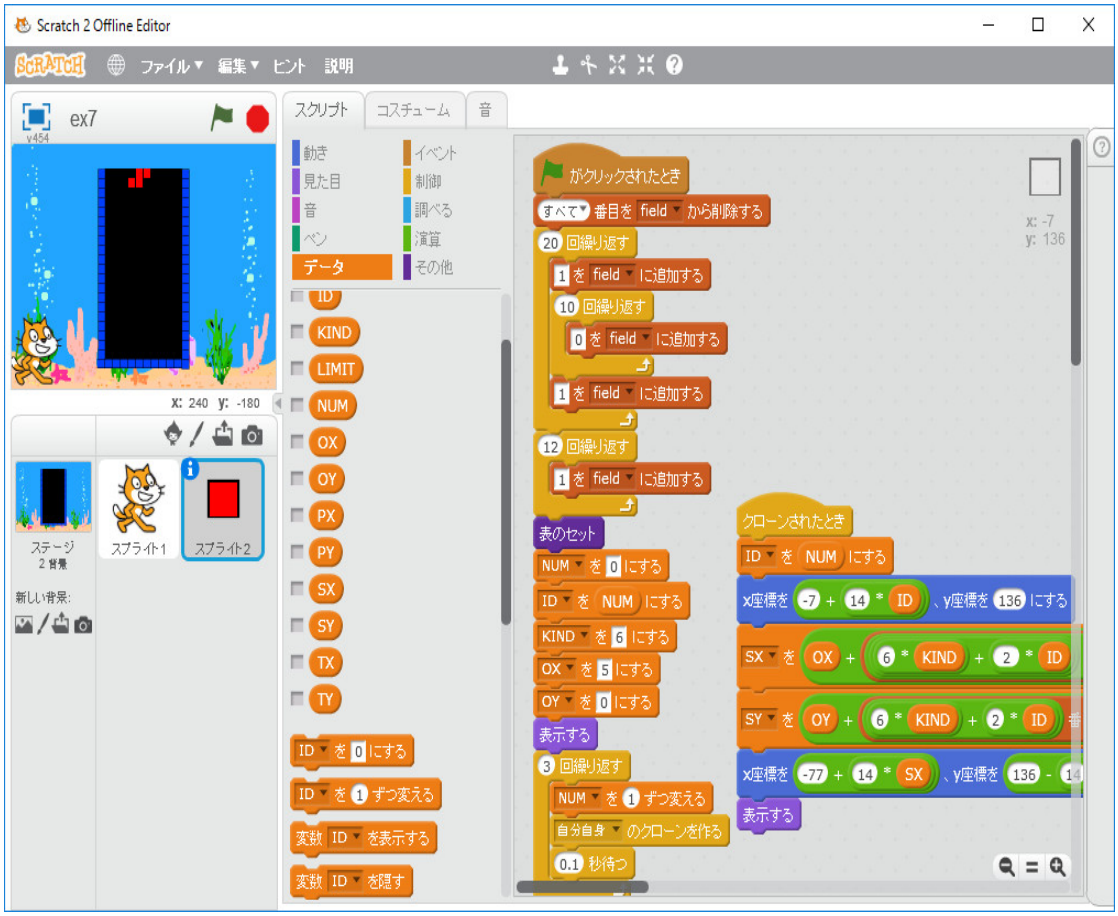
```
0 -1 0 1 0 2

-1 -1 0 -1 0 1

0 -1 -1 0 0 1

0 -1 -1 1 0 1

0 -1 1 0 1 1

0 1 1 0 1 1

0 1 1 -1 1 0
```

を table にセットすれば良いです。我々は x 座標、y 座標の順番の方がなじみがあるので、縦と横の並びを逆にして table にセットします。

プログラミングに戻ります。「すべてのスプライト用」のリストとして table を作る。引数なしのブロック「表のセット」を作成し、field のセットの直後に、ブロック「表のセット」を追加し、ブロック「表のセット」で table にデータをセットする。

tabel のデータが間違っていないか確かめましょう。

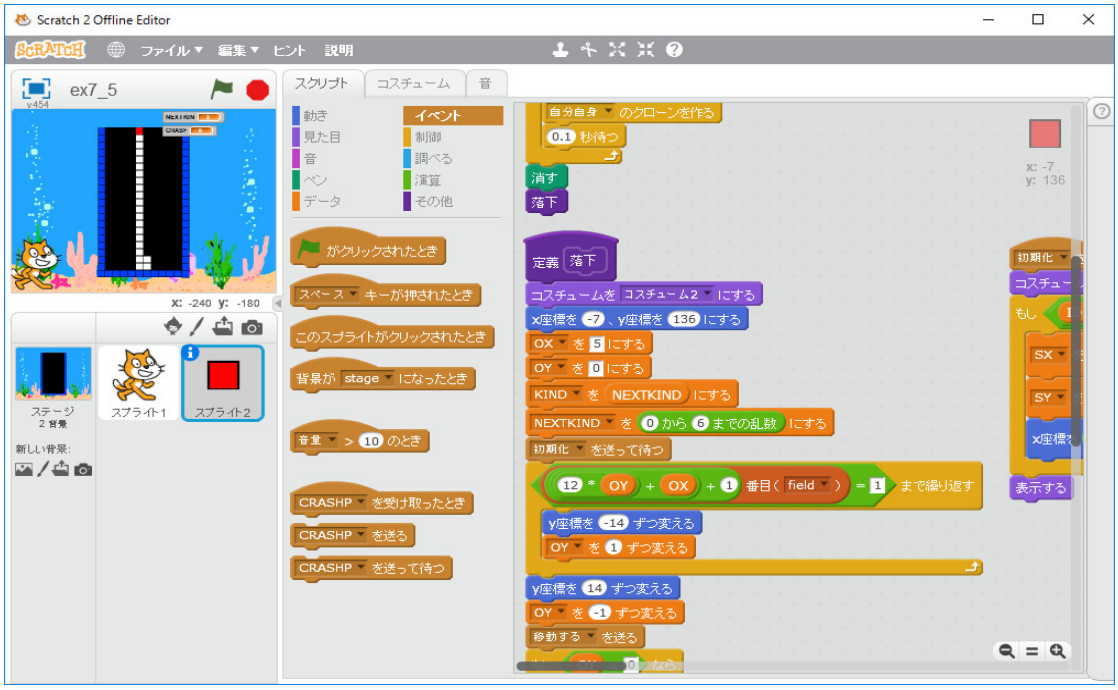
7 種類のテトリミノのどれであるかは KIND=0,1,2,3,4,5,6 で区別します。ID は 1,2,3 です。KIND の ID の相対位置の x 座標は $table[6*KIND+2*ID-1]$ にあり、y 座標は $table[6*KIND+2*ID]$ にあります。KIND の値を変えて、四個のスプライトを table のデータに基づいて表示してみましょう。



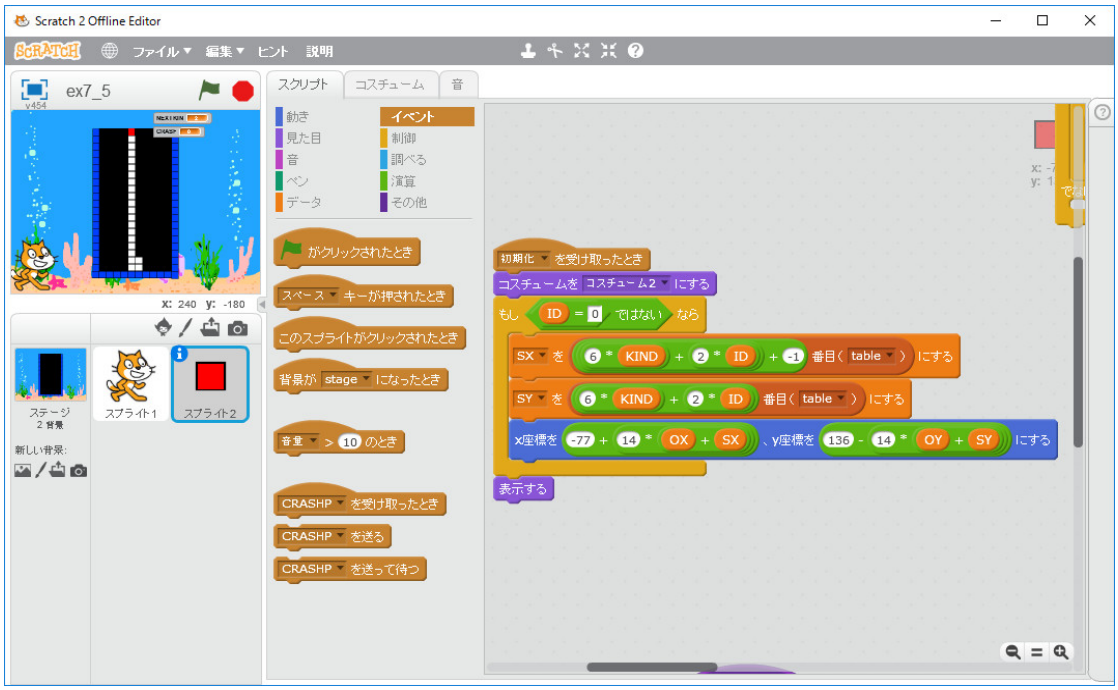
「クローンされたとき」の定義が切れていますが、何をやっているか分かると思います。table にセットしたデータが間違っていないことを確かめるためにプログラムを変更しただけで、チェックが済んだら元に戻します。

さて、「落下」の定義でオリジナルのスプライトが移動したらメッセージをクローンに送ります。次に落下するテトリミノが分かっていたら作戦を考えられるようになりますから、次のテトリミノの種類を保持する変数 NEXTKIND を作って、「旗がクリックされたとき」の定義で「0 から 6 までの乱数」でセットしておきます。

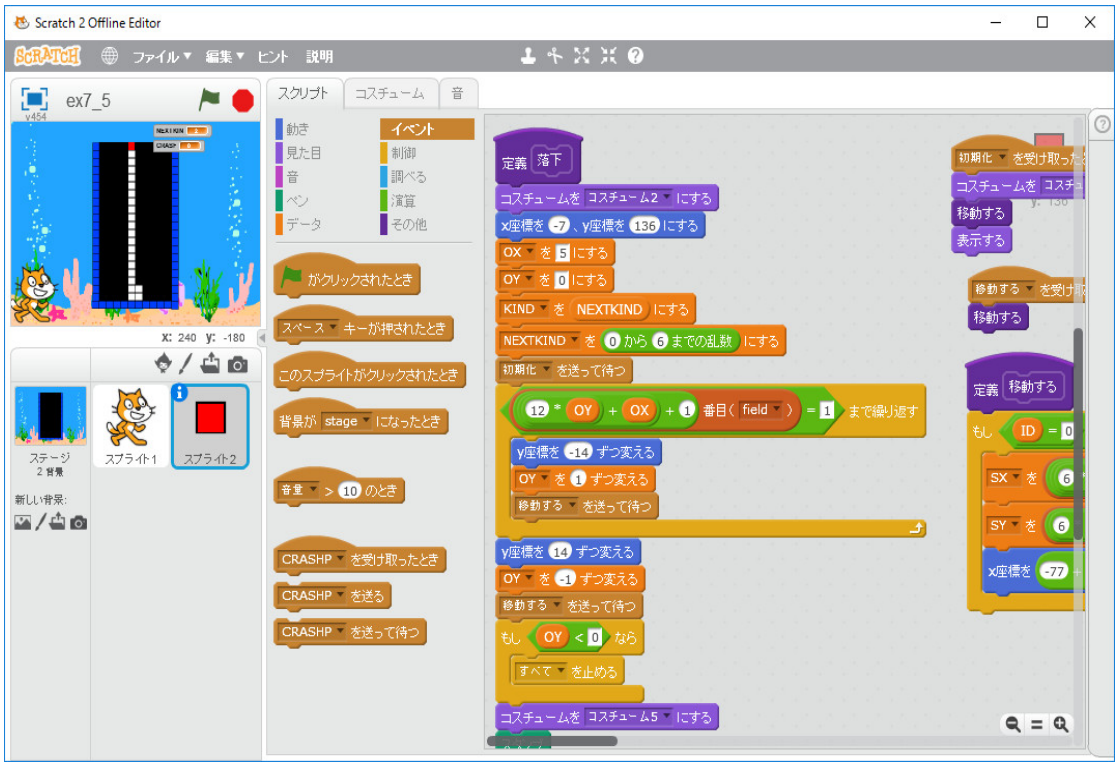
まず、「落下」の定義の初めに、オリジナルなスプライトの初期位置をセットします。KIND を NEXTKIND とし、NEXTKIND を「0 から 6 までの乱数」でセットします。オリジナルなスプライトの初期位置をセットし、フィールド座標 (OX,OY) をセットすれば、イベントの「メセージを送って待つ」のタグで新しいメセージをクリックして、「初期化」として、「初期化を送って待つ」をはめ込みます。

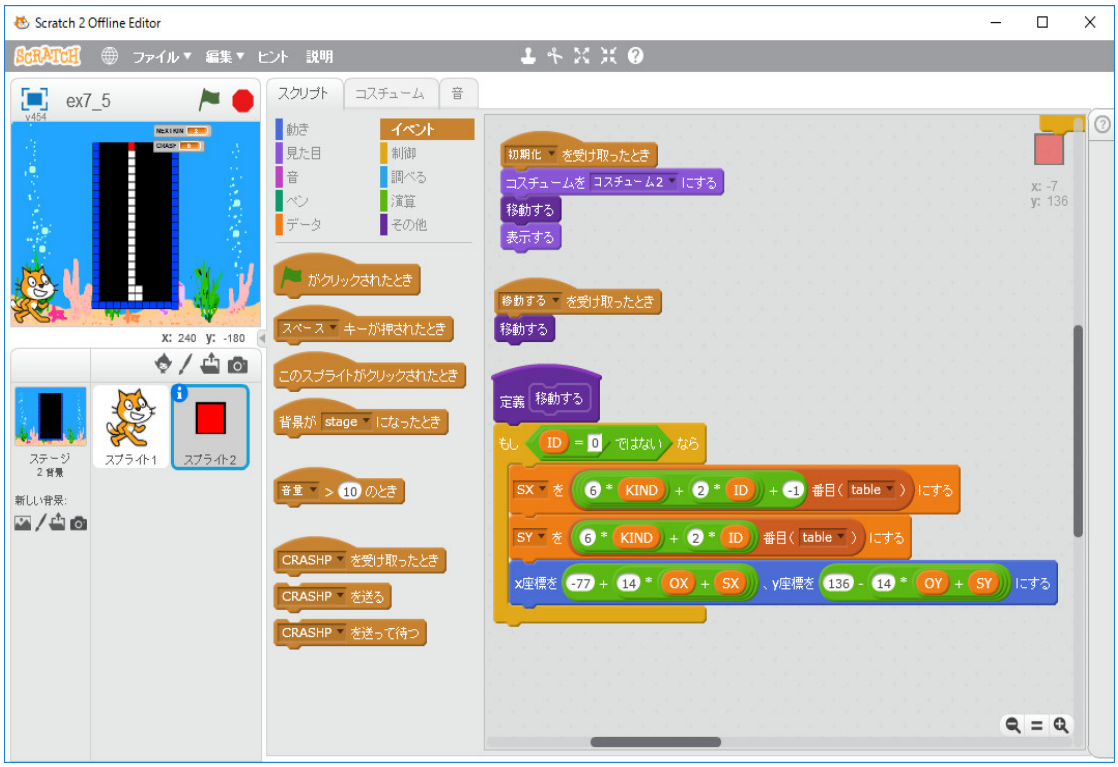


「初期化を受け取ったとき」の定義は、単純です。各クローンのローカル変数　（SX,SY）にクローンのフィールド座標に関する相対座標を保持するようにします。$SX = table[6*KIND+2*ID-1]$、$SY = table[6*KIND+2*ID]$ です。

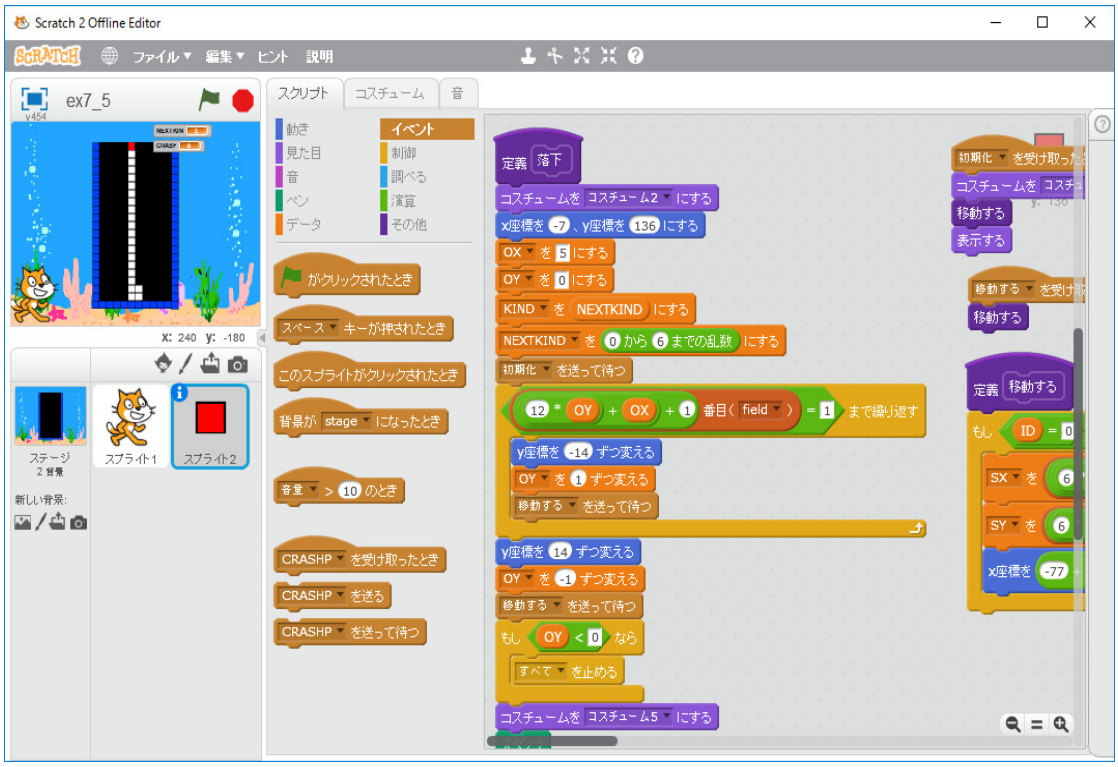


　オリジナルのスプライトが移動したら「移動」のメッセージをクローンに送ります。これは「初期化を受け取ったとき」とほとんど同じことをします。「初期化を受け取ったとき」の定義をブロック「移動する」として定義し、「初期化」のメッセージと「移動」のメッセージで共用します。

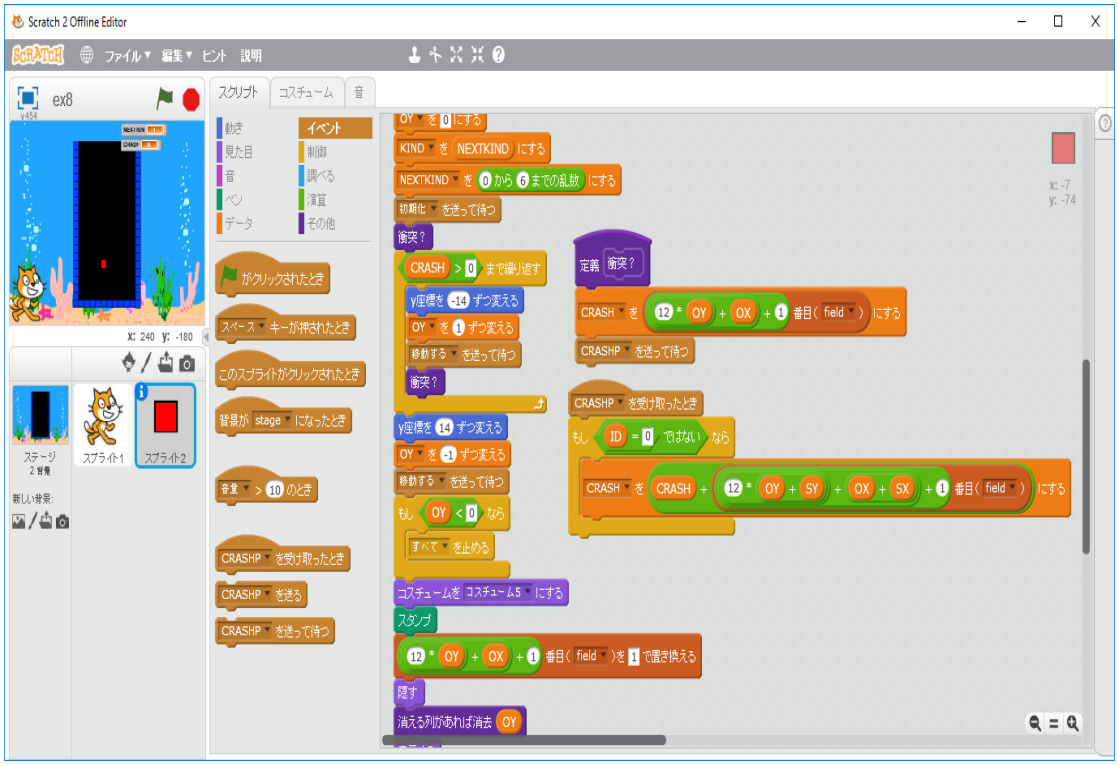

Scratch 2 Offline Editor
ファイル
編集
ヒント
説明
ex7_5
スクリプト
コスチューム
音
イベント
定義 落下
定義 移動する
ステージ
スプライト1
スプライト2


Scratch 2 Offline Editor
ファイル
編集
ヒント
説明
ex7_5
スクリプト
コスチューム
音
イベント
定義 落下
定義 移動する
ステージ
スプライト1
スプライト2

次に

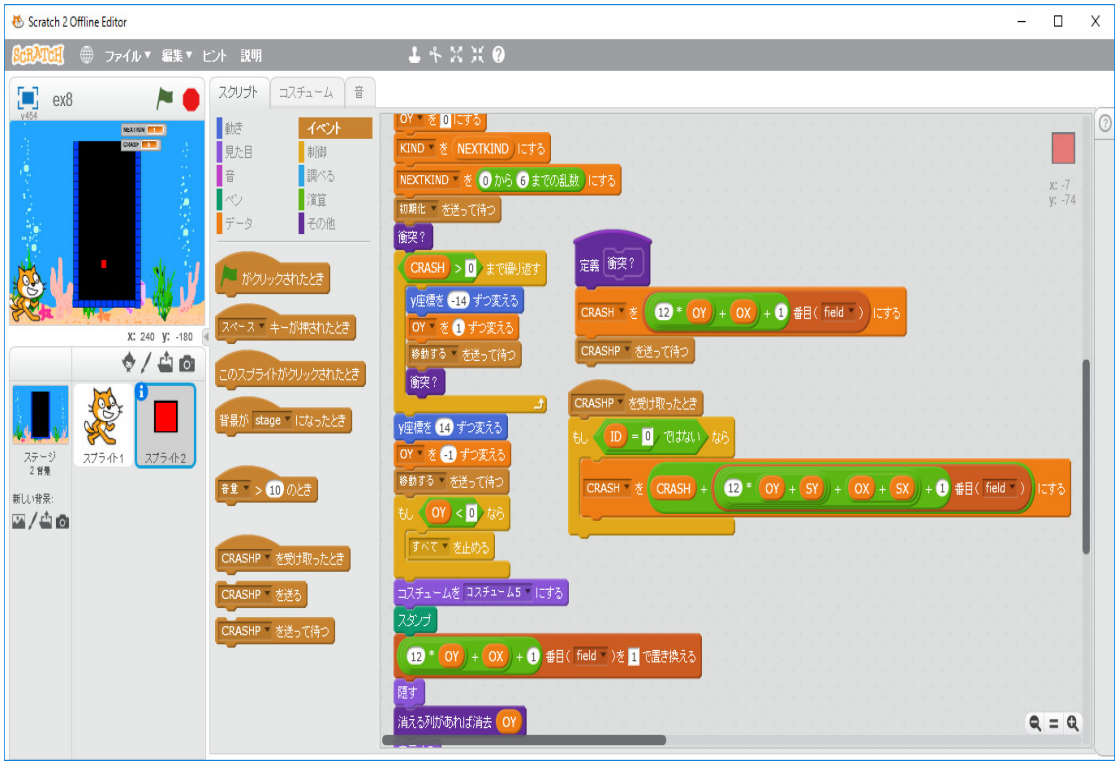


にある「(12*OY+OX+1) 番目（field）=1 まで繰り返す」を直す必要があります。これはオリジナルのスプライトの衝突しか考慮していません。クローンの衝突も考慮する必要があります。衝突したかどうかを計算した結果を保持する変数 CRASH を作る。Scratch では、関数を作ることが出来ないので、ブロックと戻し値を保持するグローバル変数 CRASH で代用します。

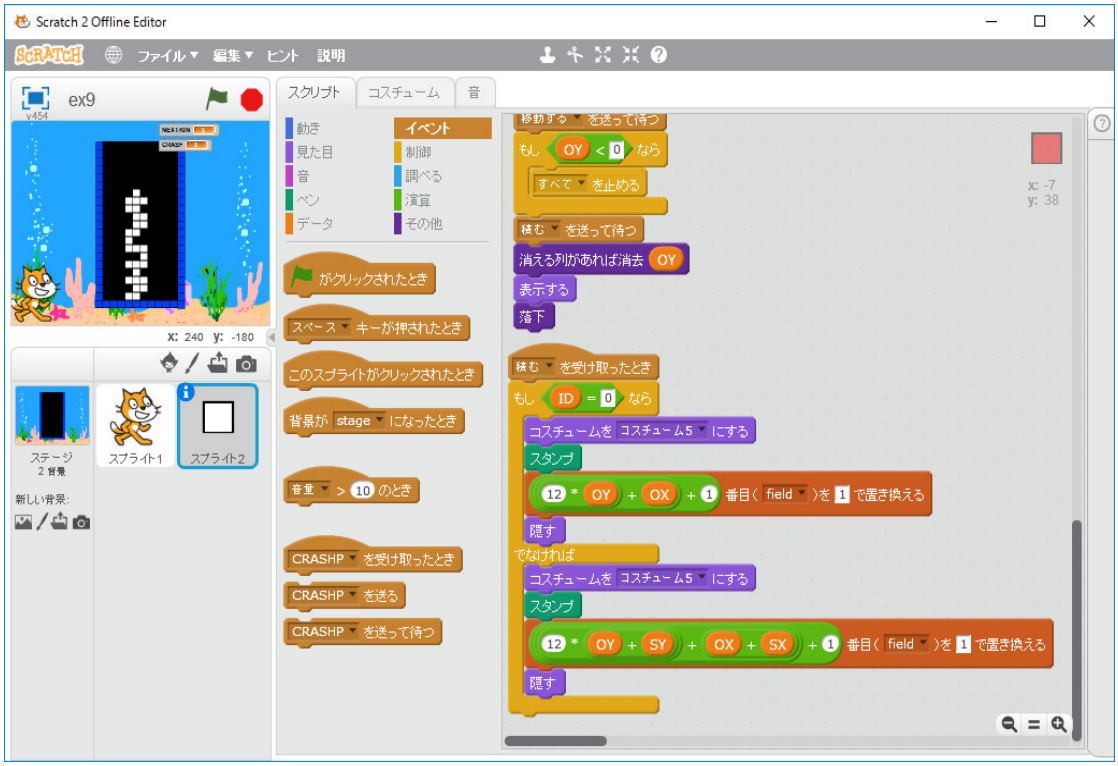


上のの図のようにブロック「衝突？」を使います。次はブロック「衝突？」の定義です。ブロック「衝突？」の中では、オリジナルのスプライトの衝突だけ考え、「CRASHP」というメセージを送り、各クローンで変数 CRASH を変更してもらいます。

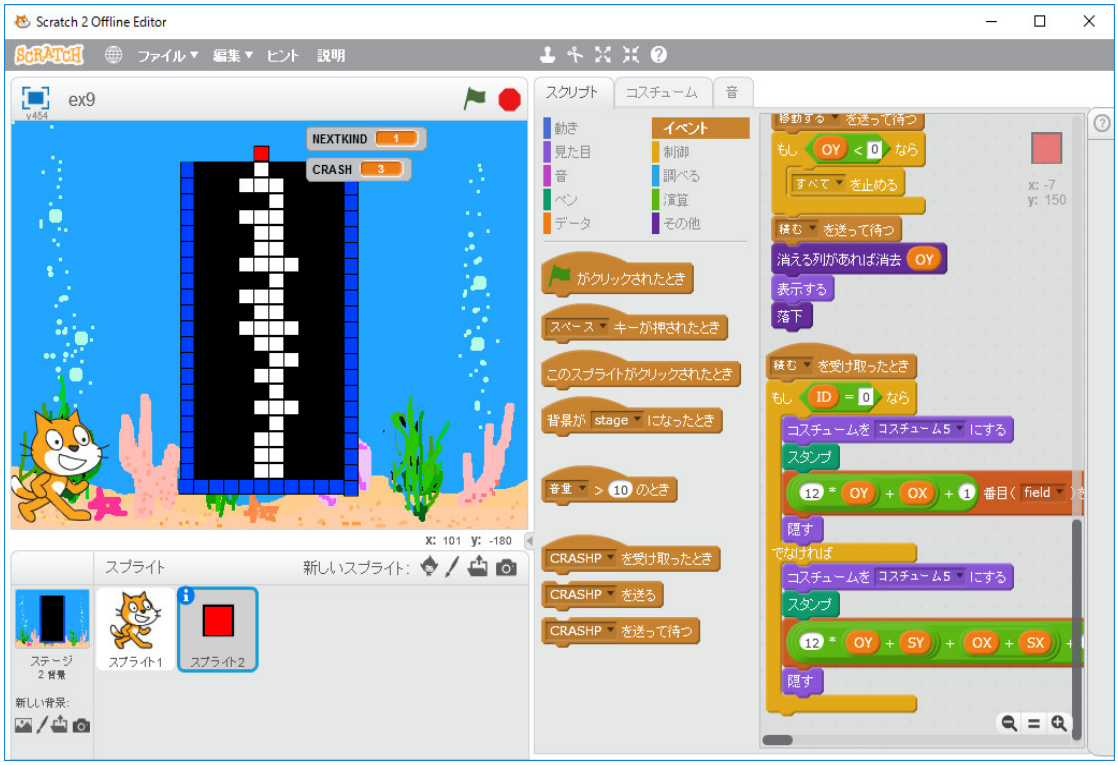
次は衝突した後の処理です。



では、オリジナルのスプライトしか積みあがりません。クローンも積みあがるようにします。「コスチュームをコスチューム５にする」からの４行をとクローンの同様な処理を「積む」というメセージを送ることで実現します。

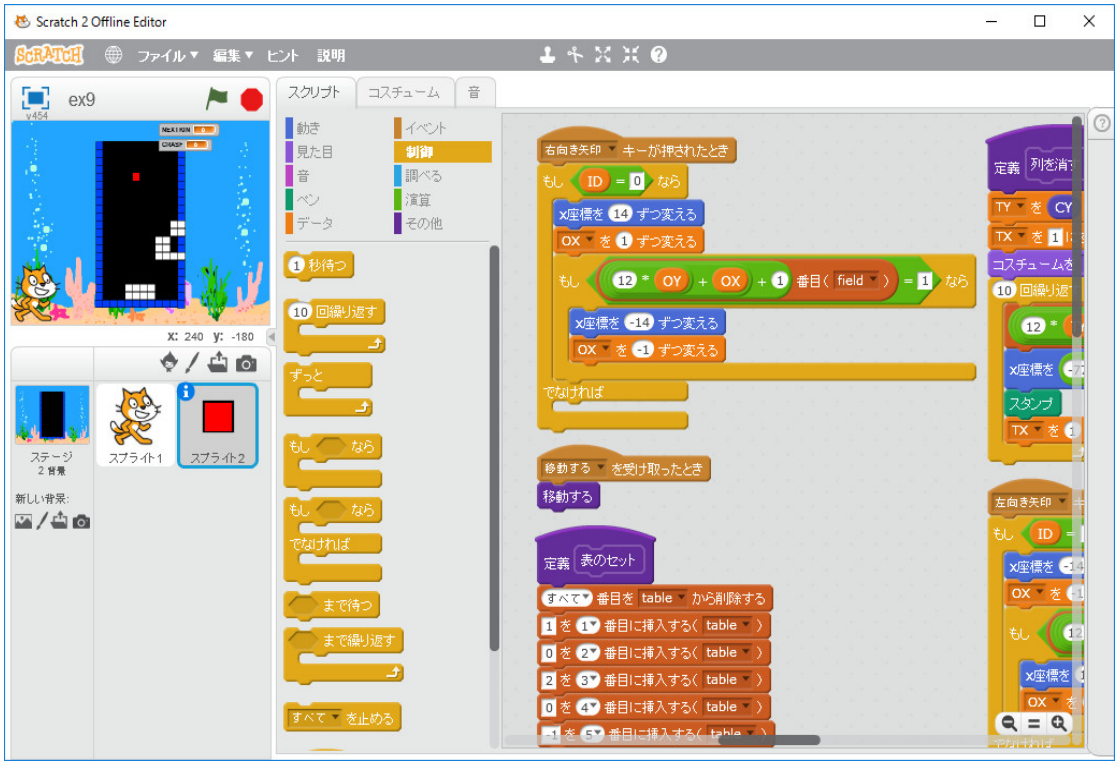


実行してみます。

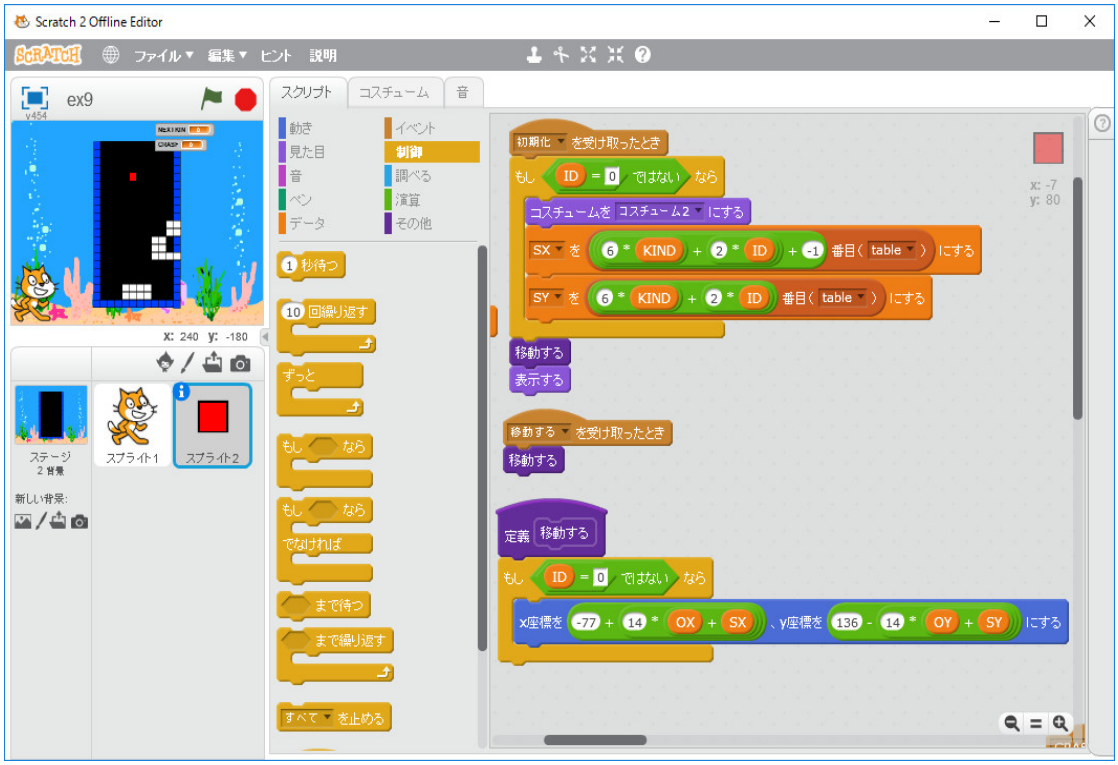


間違っています。

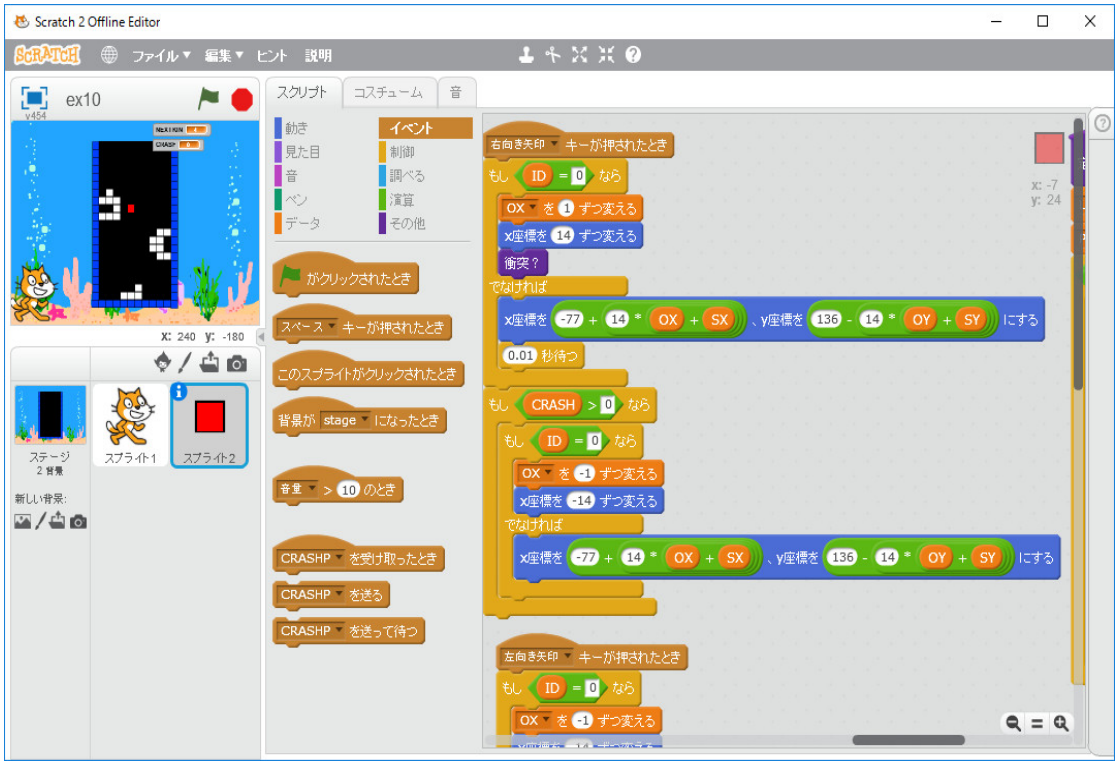
右に移動した時、エラーになります。原因は「右向き矢印キーが押されたとき」と「左向き矢印キーが押されたとき」のクローンの処理を定義していなかったことです。



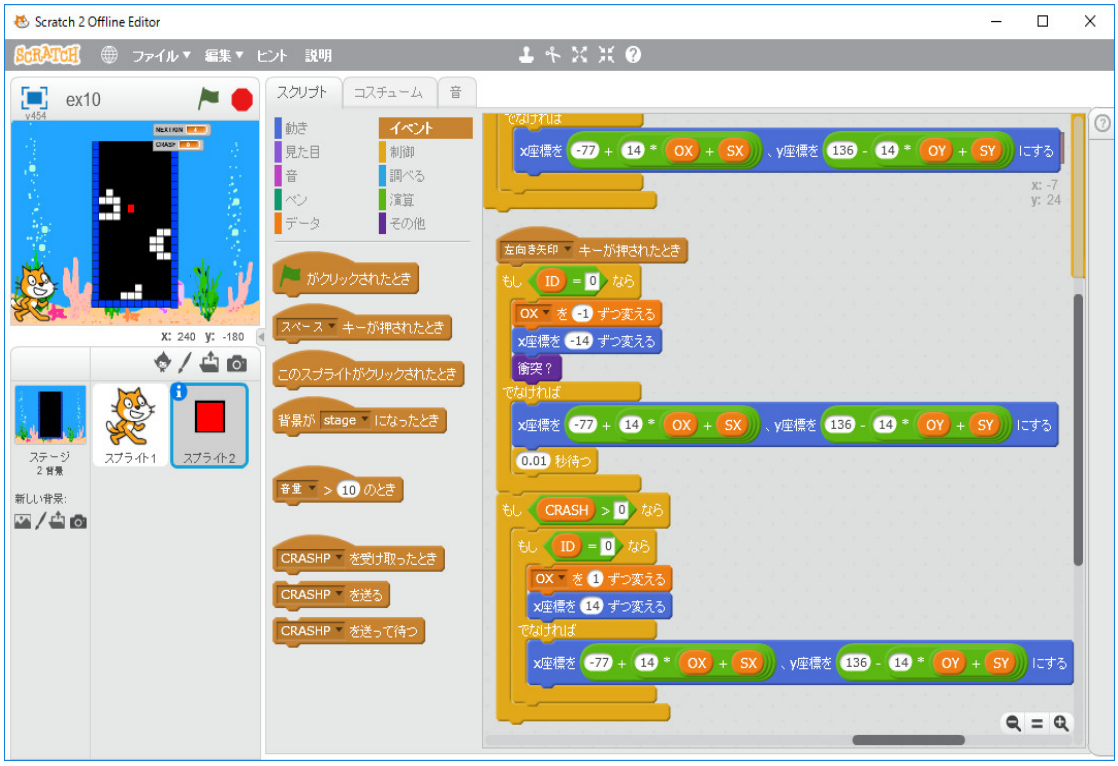
「右向き矢印キーが押されたとき」と「左向き矢印キーが押されたとき」のクローンの処理を定義します。その前に、「初期化を受け取ったとき」の定義とブロック「移動する」の定義を下図のように変えておきます。ブロック「移動する」は下に移動するだけだから、クローンを下に移動するだけでよい。回転を考えた時、今までの定義ではエラーになる。



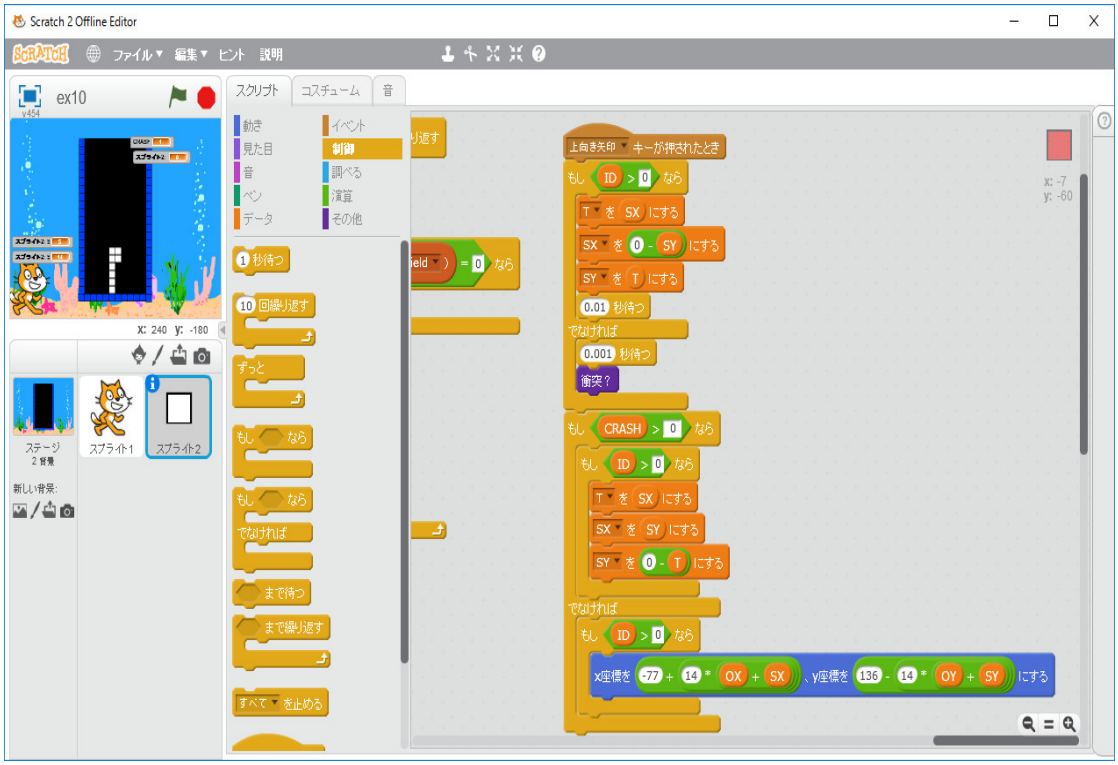
「右向き矢印キーが押されたとき」の定義は下図のようにすればいい。



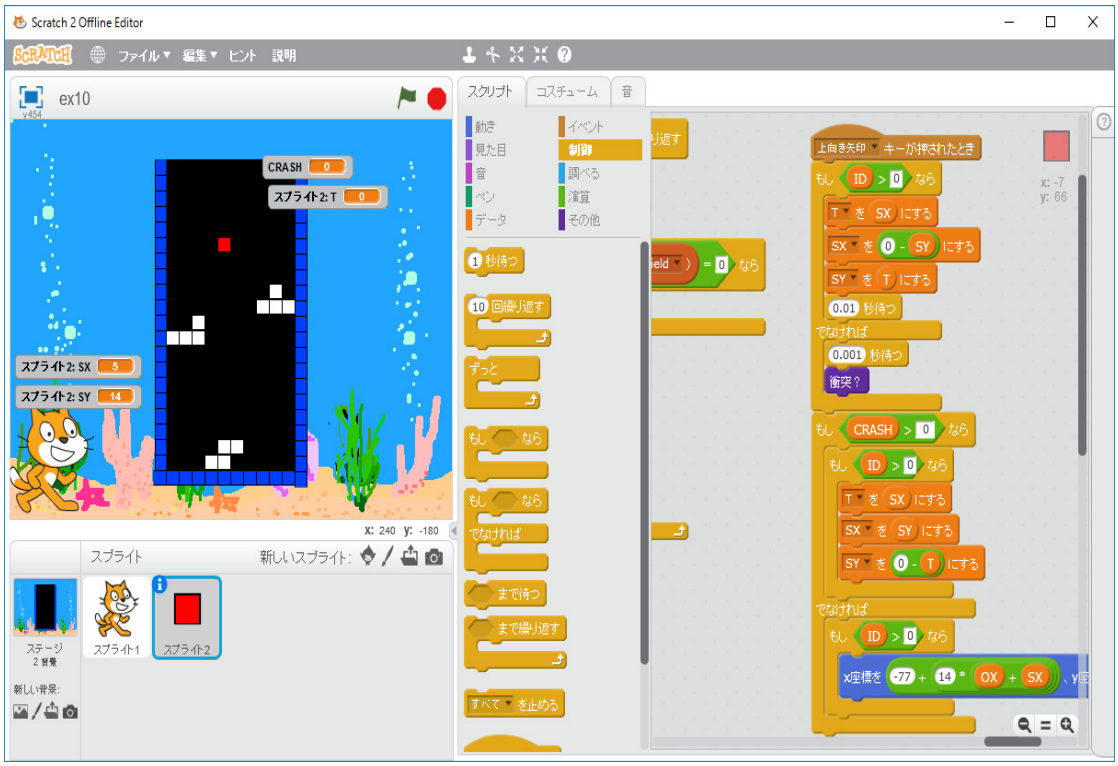
「左向き矢印キーが押されたとき」の定義も同様で下図のようにすればいい。



「上向き矢印キーが押されたとき」に回転するようにしよう。SX と SY の値を符号を変えて入れ替えるために、補助変数 T を作っておく。

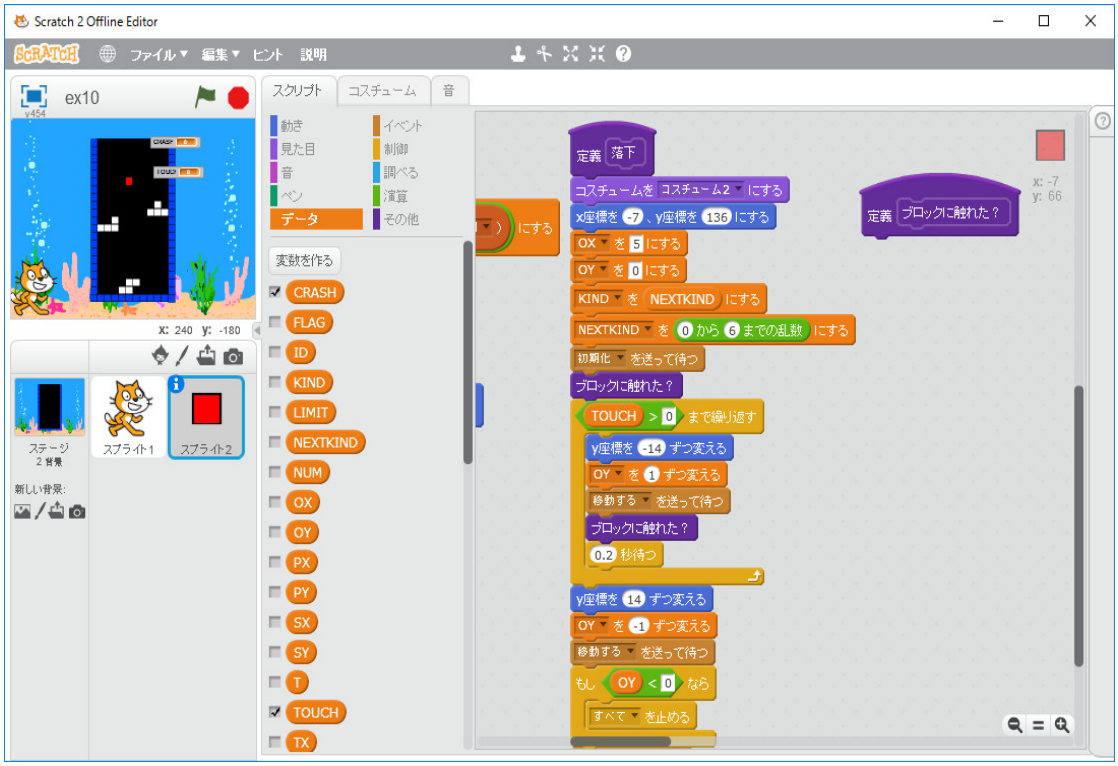


実行してみる。間違っています。



「落下」の時の「衝突？」と「右向き矢印キーが押されたとき」や「左向き矢印キーが押された

とき」の「衝突？」は同じものを使ってはダメです。色々のことがブラックボックスになっているので正確なことは分かりませんが、多分スプライトのプログラムが同時にいくつも実行されているので、壁にめり込んでいる間に落下し、「落下」のプログラムでも衝突していると判断され、「積む」というメセージが送られていると思います。「落下」の時の「衝突？」は壁に衝突しても「積む」というメセージを送ってはいけません。対策は壁に食い込まずに衝突の判定をすべきだと思います。「落下」の時の「衝突？」を壁にめり込んだのは衝突と判定しないブロック「ブロックに触れた？」として作ることにします。変数 CRASH も専用の変数 TOUCH を使います。

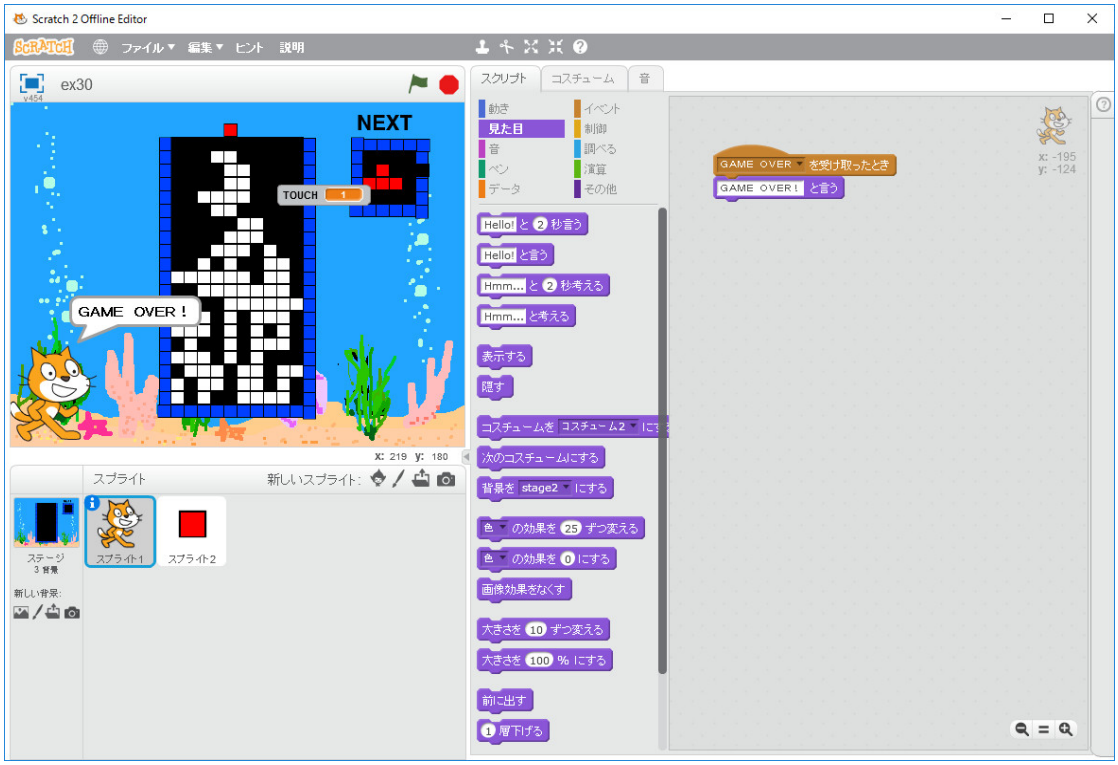


ブロック「ブロックに触れた？」は、メッセージ「TOUCHP」を作り、

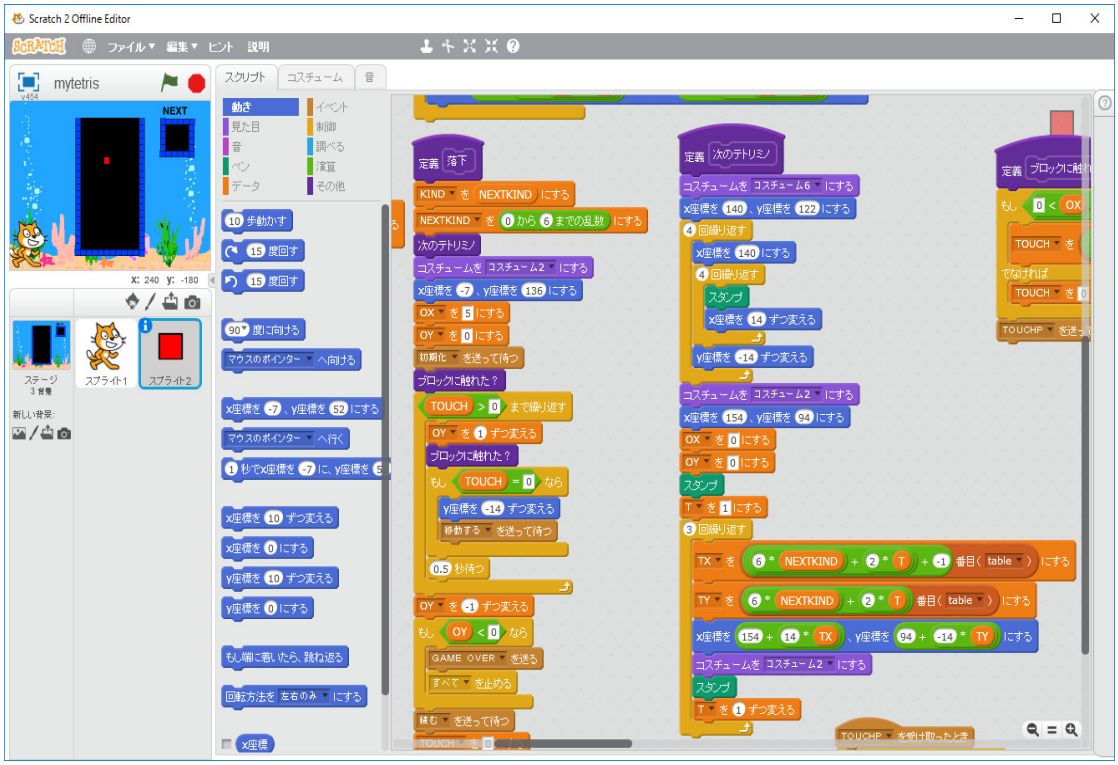
のようにプログラミングします。

次に次に現れるテトリミノを表示するようにしましょう。そのためには次に現れるテトリミノを表示する場所を背景に作って置かなければならない。この背景を作ったプログラムを修正してそのような背景をつくります。次に現れるテトリミノを表示するようにします。プログラムは自分で考えて下さい。もうできるはずです。

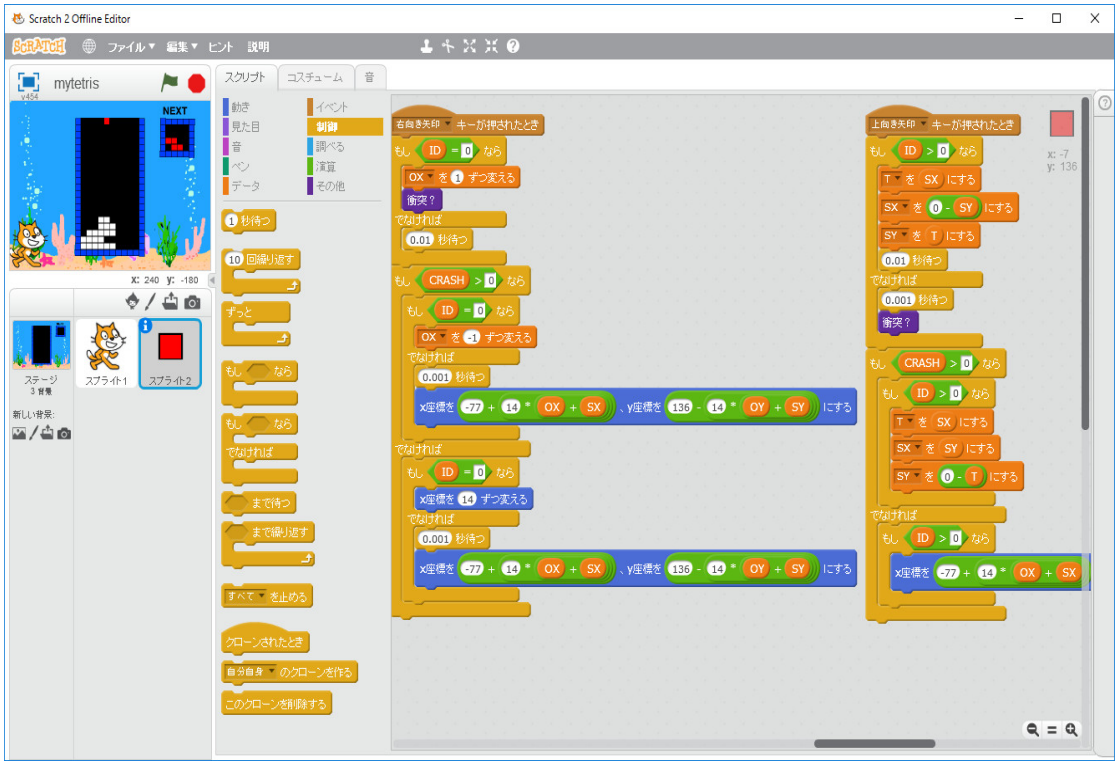
ゲームオバーになれば、「gameover」のメッセージを送り、猫に「GAME OVER」と言わせます。

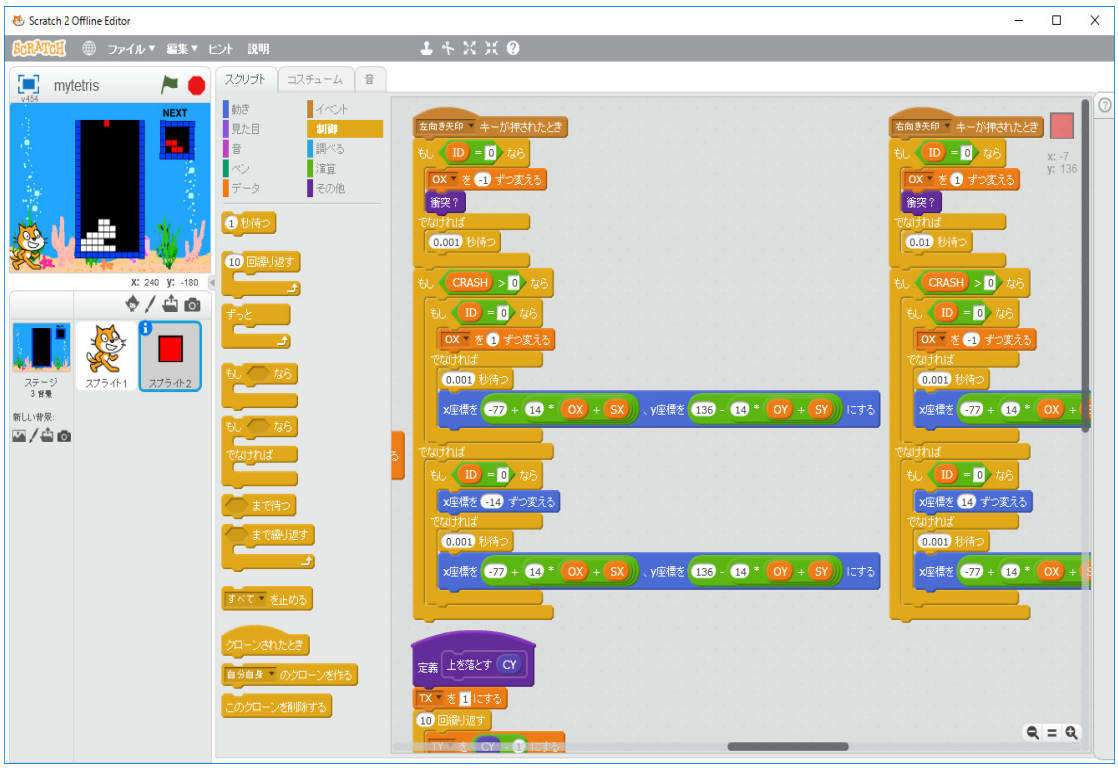


テトリミノが底や積まれたテトリミノに乗っかってから上昇しているのが見えるので、それを直してみます。「落下」のブロックの「$TOUCH > 0$ まで繰り返す」ループの中のオリジナルの四角の表示位置を変える命令「y 座標を -14 ずつ変える」を底やブロックに触れないことが判明した後表示するよう次の図のように手直しします。

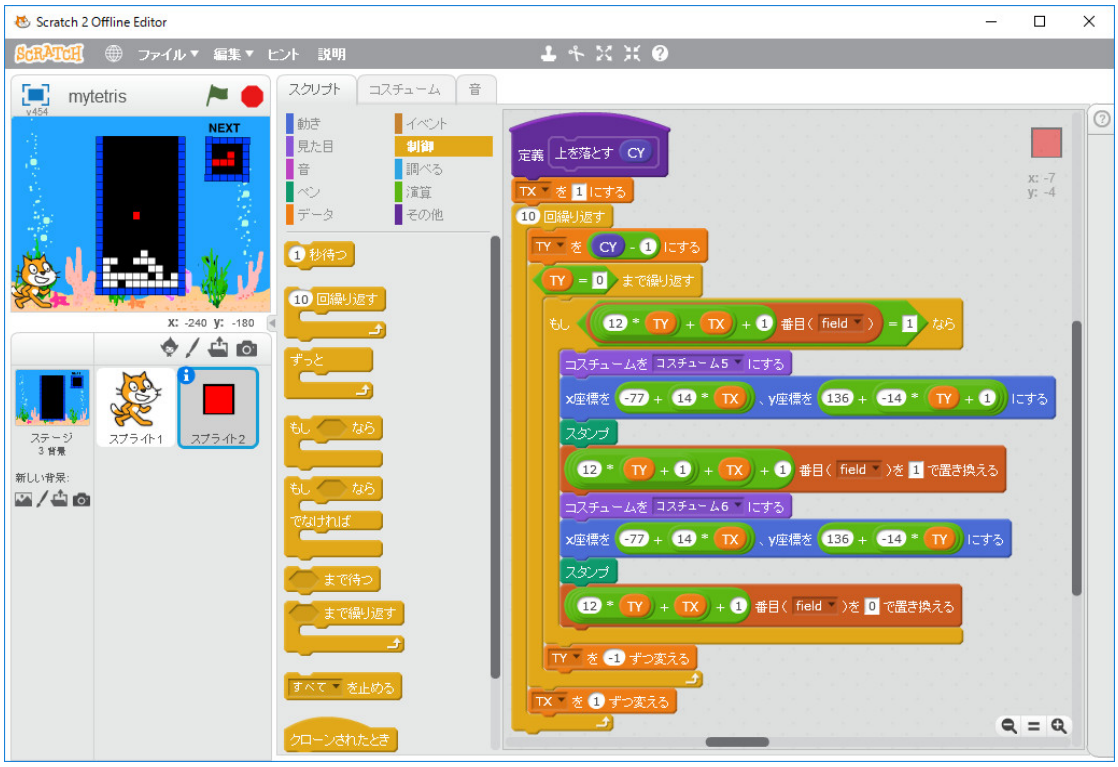


更に、「右方向矢印キーが押されたとき」と「左方向矢印キーが押されたとき」を次のように手直しします。

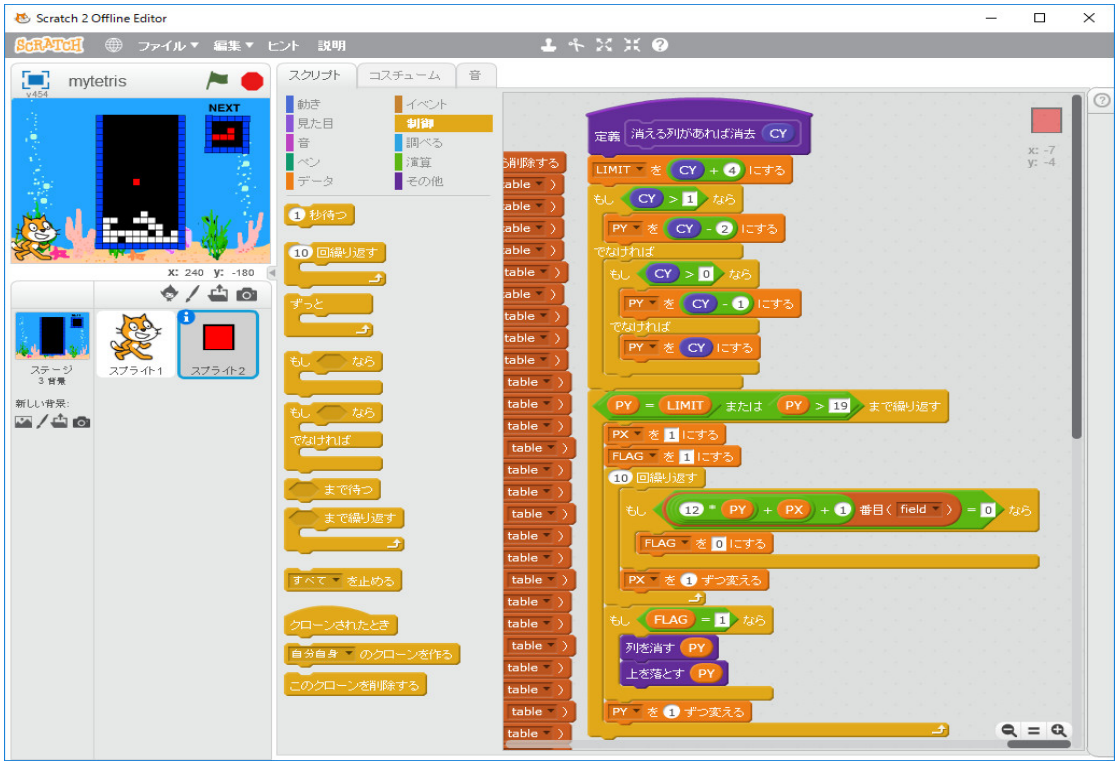




プログラムが間違っています。ブロック「上を落とす」が間違っています。



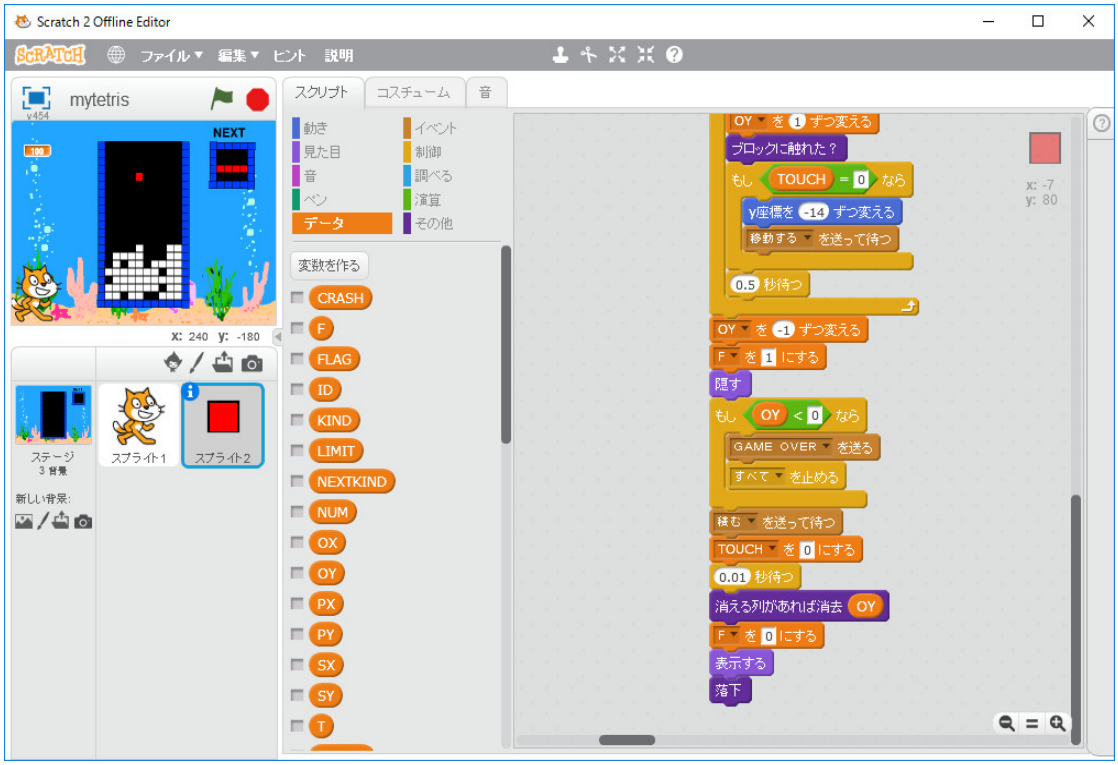
と修正します。ブロック「消える列があれば消去」も間違っています。



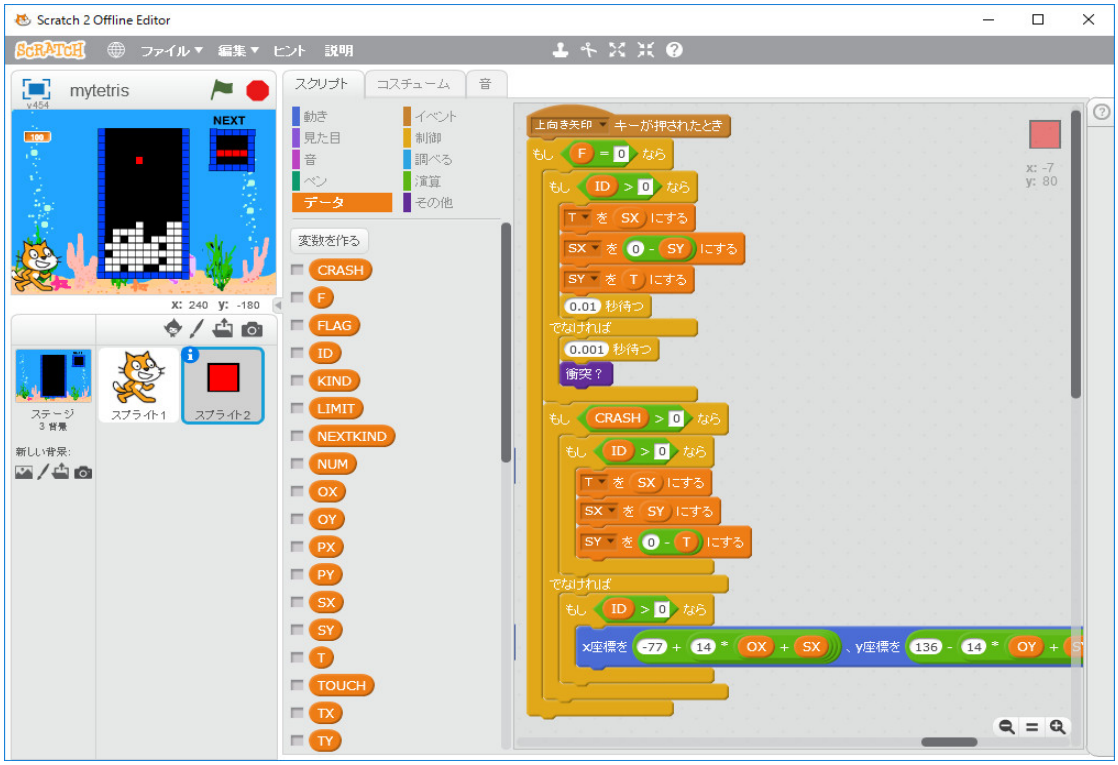
と修正します。これらのブロックは一個の四角形の落下なら正しかったのですが、テトリミノが落

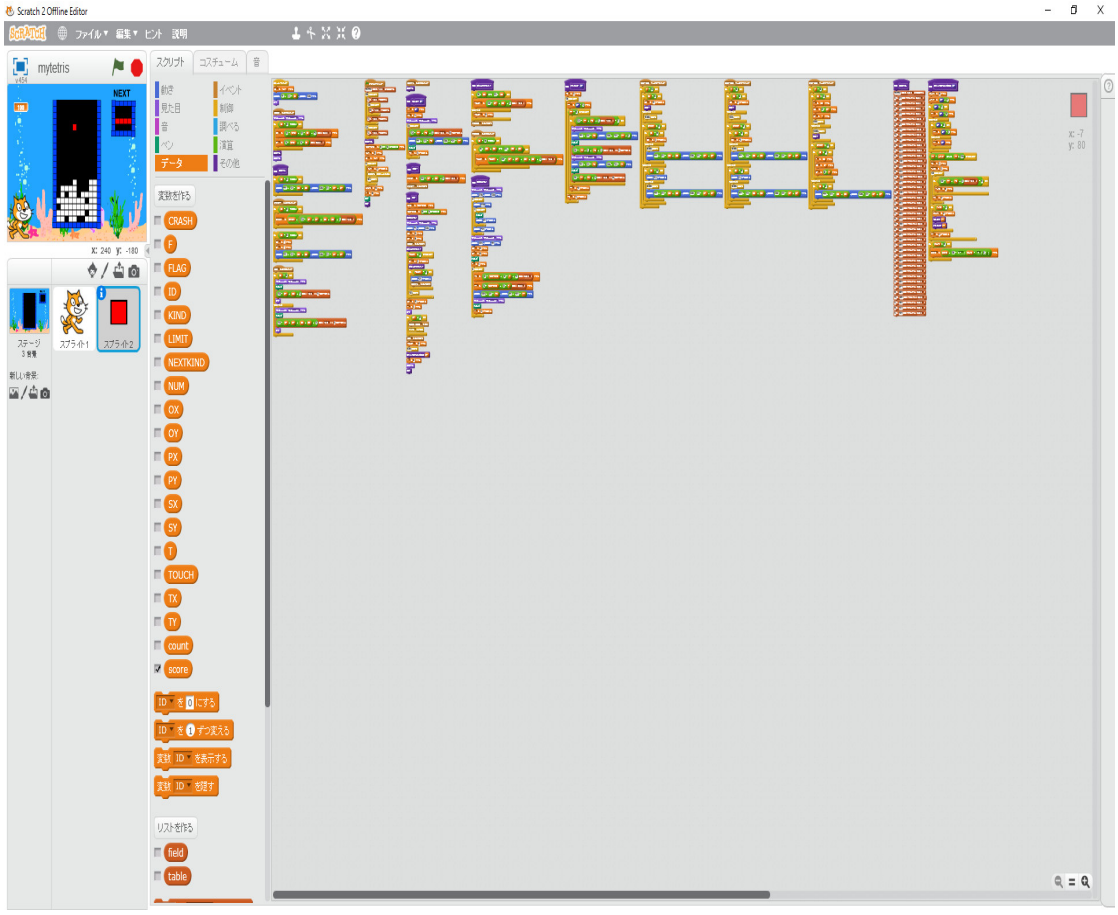
下するようになると間違っています。このような間違いはこのような段階的なプログラムの作り方をするとよく起こります。自分で間違いに気づき（ゲームのプログラムのような視覚的な結果の出るものでは、何処かにエラーがあることの発見は容易ですが、間違えていることに自分ではなかなか気づかないことも多いです）、自分でそれを修正できるようになったら、プログラムが一人で作れるようになります。これが一番時間がかかる辛い作業で、ここで行き詰ってプログラミングを諦める人が多いですが、何とか乗り越えてください。私はプログラムが出来たと思ってから、実際に出来上がるまで、それまでかかった 2 倍の時間が必要なことが良くあります。

最後に、衝突の処理を始めたらフラグ F を立てて、矢印キーを無効にする処理を付け加えます。つまり、グローバル変数 F を作り、「旗がクリックされたとき」に F = 0 とセットし、ブロック「落下」で「ブロックに触れた？」で $TOUCH > 0$ となると、F=1 とし、F=1 の間は矢印キーを無効にします。ブロック「消える列があれば消去」が終われば、F=0 に戻し、矢印キーを有効にします。



矢印キーが押されたときの処理は、下図のように「もし F=0 なら」で全体を囲みます。

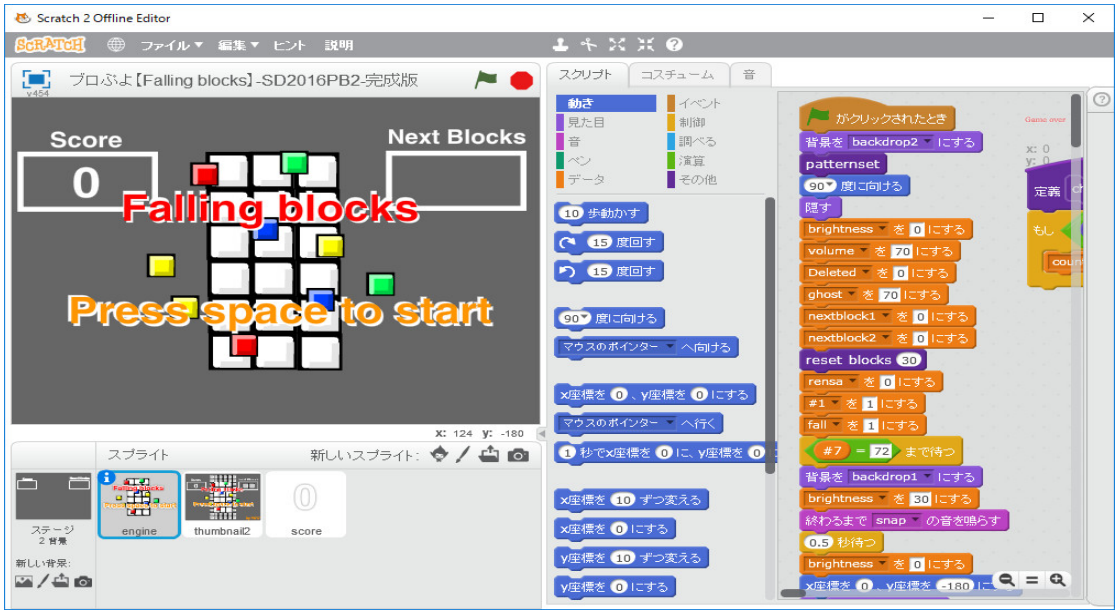

Scratch 2 Offline Editor
ファイル
編集
ヒント
説明
mytetris
NEXT
スクリプト
コスチューム
音
動き
見た目
音
ペン
データ
イベント
制御
調べる
演算
その他
変数を作る
CRASH
F
FLAG
ID
KIND
LIMIT
NEXTKIND
NUM
OX
OY
PX
PY
SX
SY
T
TOUCH
TX
TY
上向き矢印 キーが押されたとき
もし F = 0 なら
もし ID > 0 なら
T を SX にする
SX を 0 - SY にする
SY を T にする
0.01 秒待つ
でなければ
0.001 秒待つ
衝突？
もし CRASH > 0 なら
もし ID > 0 なら
T を SX にする
SX を SY にする
SY を 0 - T にする
でなければ
もし ID > 0 なら
x座標を -77 + 14 * OX + SX 、y座標を 136 - 14 * OY +
ステージ
3 背景
スプライト1
スプライト2
新しい背景:


Scratch 2 Offline Editor
mytetris
スクリプト
コスチューム
音
変数を作る
CRASH
F
FLAG
ID
KIND
LIMIT
NEXTKIND
NUM
OX
OY
PX
PY
SX
SY
T
TOUCH
TX
TY
count
score
リストを作る
field
table

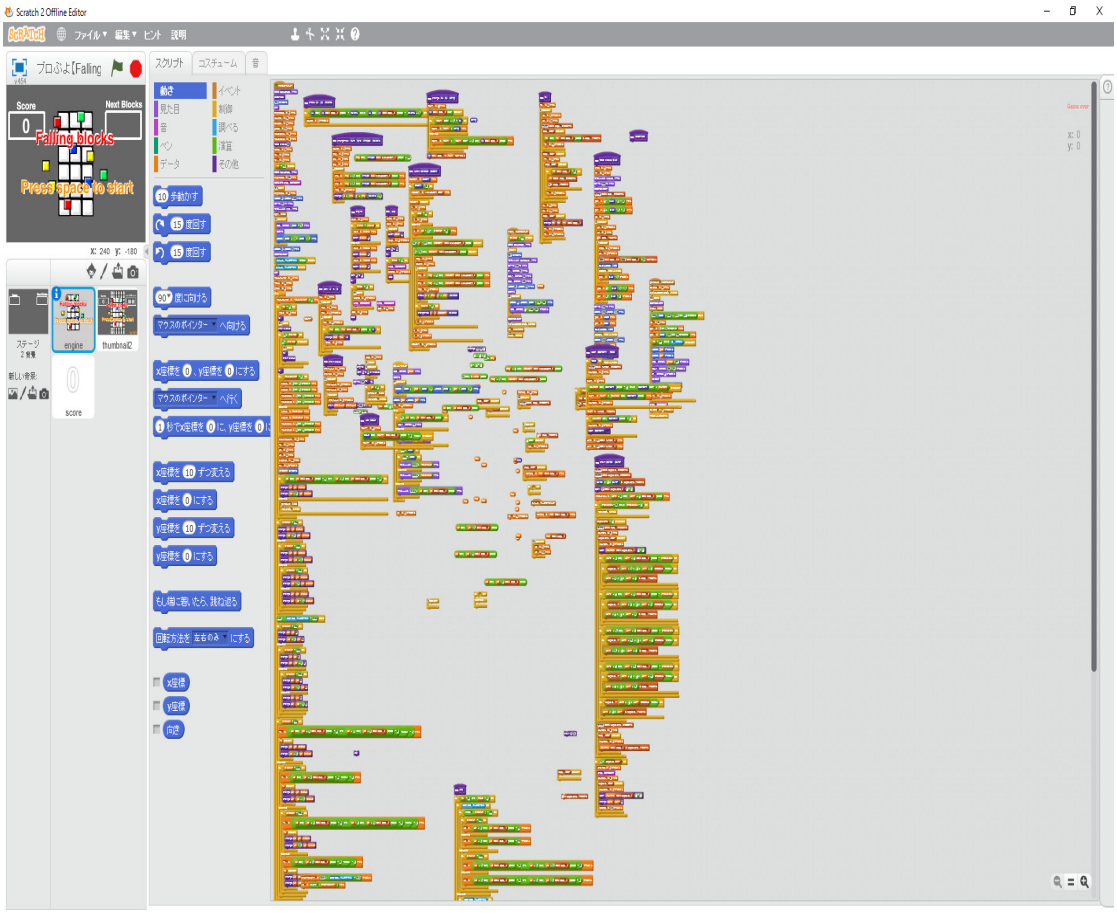
これが全体のプログラムです。いくつものスクリプトが同時に動いいるので、制御が難しいですが、落下速度を早くしたり、変なことをしなければ、多分正しく動いていると思います。プログラムが大きくなるとどこに何を書いているか分からなくなり、探し回らなくてはならず、それに時間が取られます。プログラムが印刷できないので、大きいプログラムを作るのは大変です。兎も角、それらしいものが出来ました。プログラムの断片をお示しするにも絵で表現しなければいけないので、細部がよくわからなかったかもしれませんが、そこは自分で考えるか、適当な参考書を片っ端から読んでみるか、インターネットで調べるか、仲間であっでもないこうでもないと議論してみるか、誰かよく知っている人に聞いてください。高知大学教育学部では、私が作ったプログラムですから、多分、私が一番詳しいですが、私は非常勤講師で、自分の研究室がないので、授業の時間しか大学にいません。

後はスコアを計算して、表示するようにすることですが、それは自分でしてください。何段消えたかとかどれだけの時間持ったかなどでスコアを決めるみたいです。

次に、探索のアルゴリズムの理解が必要な「ぷよぷよ」のプログラミングに挑戦してみます。「ぷよぷよ」もインターネットで探せば、例えば



のようなプログラムをゲットすることが出来、遊ぶのが目的ならこれで目標は達成です。どんなプログラムを書けばいいか知りたければ、このプログラムを研究すればいいです。他人の書いたプログラムを読んでその思考を読み取るのが得意ならですけど。



がプログラムの大半です。他人のプログラムを読んで理解するのは、私にとっては大変です。私には眺めただけではよくわからないところが色々ありました。そもそも「ぷよぷよ」というゲームがあるということは知っていましたが、「ぷよぷよ」で遊んだことが今までありませんでした。このプログラムで遊んでみて、インターネットでルールを調べて、「ぷよぷよ」というのはどのようなゲームなのかを知りました。「基本的なルール」と「対戦ルール」というのがあるみたいです。テトリスのプログラムが作れたので、基本ルールの「ぷよぷよ」のプログラムも自力で作れるはずです。やってみましょう。

まずは「ぷよぷよ」とはどのようなゲームか知ることから始めます。インターネットで調べると

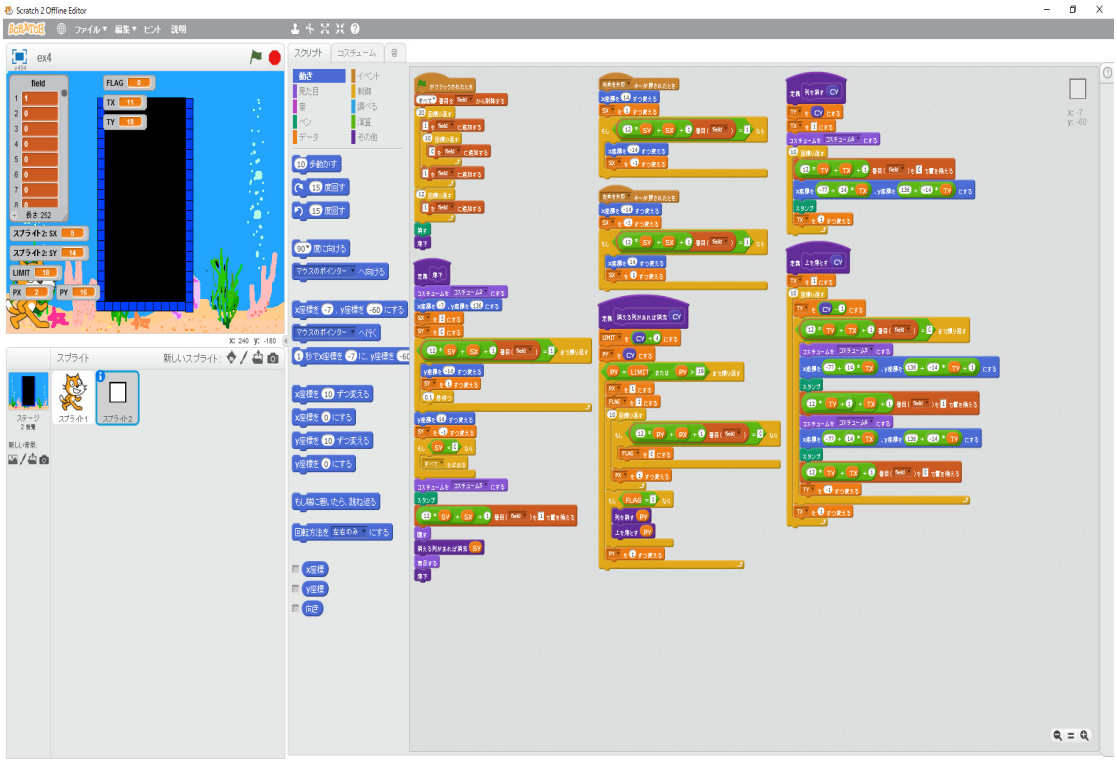
基本的なルール [編集]

「各作品によってルールの細部は異なる。ここでは基本的なルールのみを記す。フィールドは縦 12 マス×横 6 マスの格子で構成される。格子の 1 マスにつき 1 個のブロック（ぷよぷよ略してぷよ）を置くことができる。ただし、上方向は、画面外に 1 マス分だけぷよを置くことができる。上からぷよが 2 つ 1 組で落下してくる（「組ぷよ」と呼ばれる [注 4]）。ぷよは種類ごとに色が異なり、色は 3-5 色（通常は 4 色）ある。プレイヤーはぷよに対して回転、横移動、高速落下のいずれかの操作を行う。次に落下するぷよはフィールドの枠外に NEXT ぷよとして予告される。配られるぷよの配分は麻雀のツモに例えられている。落下してきたぷよがフィールドの床やほかのぷよに衝突すると、その位置にぷよが固定される。ただし、組ぷよを横にして置いたりなどして、ぷよに 1 マス分でも下方向に空白がある場合は、強制的にそのぷよだけ落下する。固定されたぷよと同色の’ぷよ’が周囲 4 方向にいる場合、それらは互いにくっつく。ぷよが 4 個以上くっつくと消滅し得

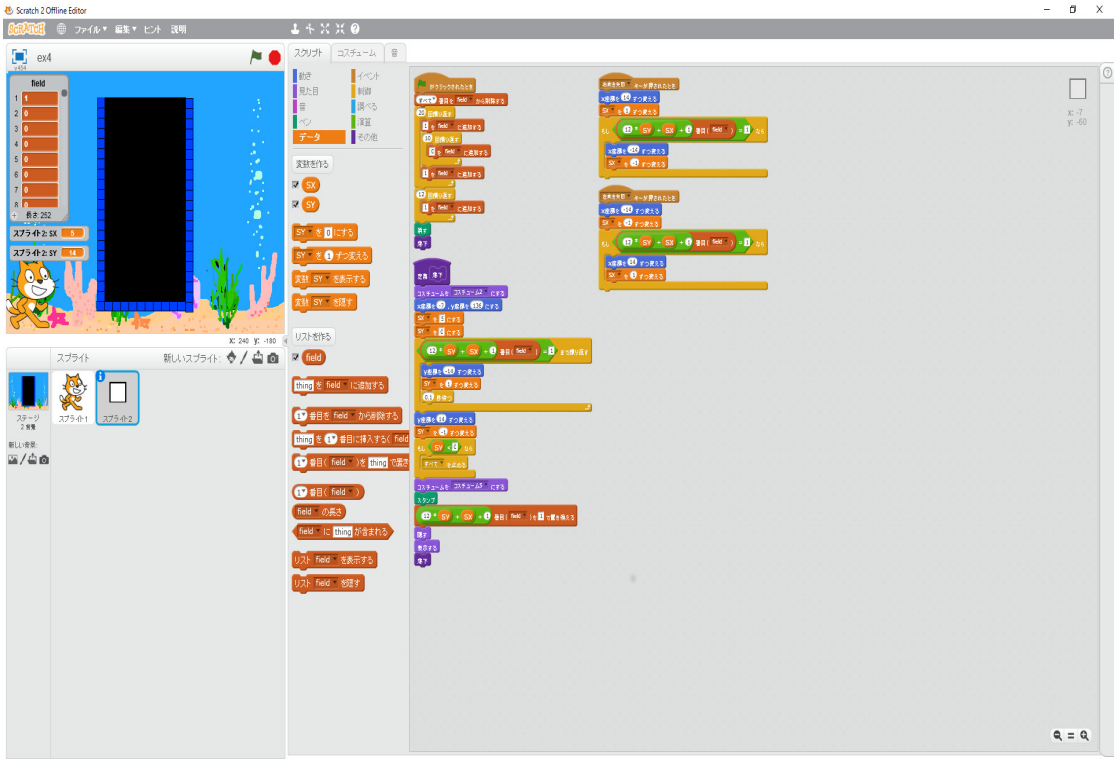
点となる。ぷよの消滅により上にあった’ぷよ’が落下する。このとき再びぷよが 4 個以上くっつくと消滅し、連鎖が起きる。なお、普通に 4 つ色を並べて消す行為だけでも 1 連鎖と考え、消滅した回数（〇回）に応じて〇連鎖と呼ばれる。複数色を同時に消した場合でも、1 連鎖扱いとなる。ぷよを消したときに入る得点は、消したぷよの数に、設定された「連鎖倍率」を掛けることで計算できる。左から 3 列目が一番上まで埋まると”窒息して”ゲームオーバー。」

と書いてありました。対戦ルールも説明していますが複雑なので今回は考えません。対戦ルールで、2 人の人がプログラミンして（何をプログラミングしているか理解できませんでしたが）、プログラミングの技能を競う大会の模様を youtube で見たことがあります。凄く奥の深いゲームみたいです。

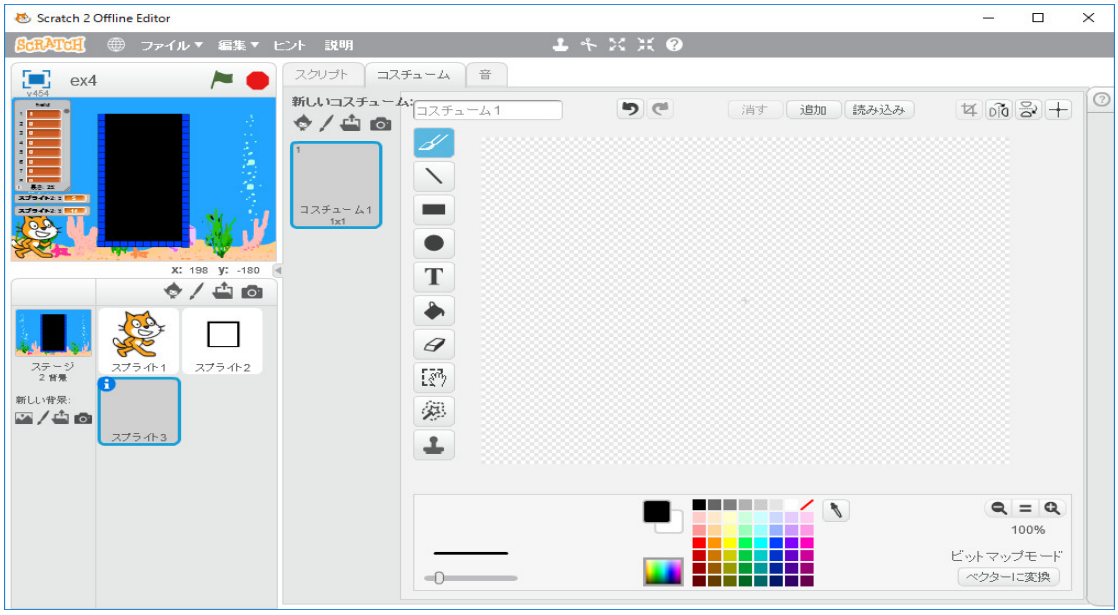
始めましょう。ゼロから始めるのはもったいないので、テトリスの作成の初めの方で作ったプログラムを保存していたのでそれを修正しながらプログラミングしていきます。



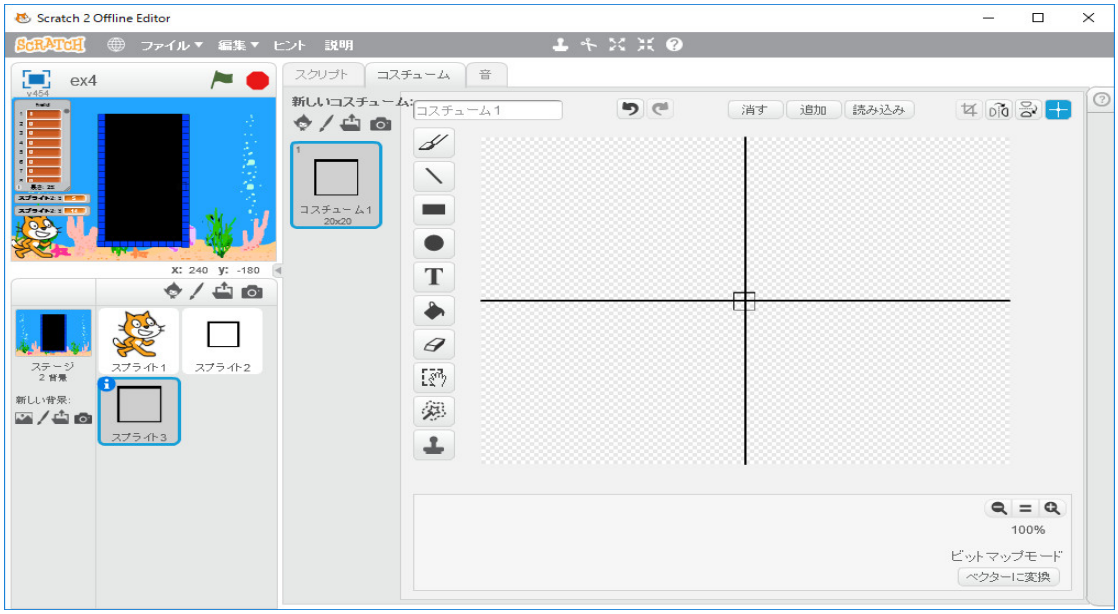
がそのプログラムです。「消える列があれば消去」、「列を消す」、「上を落とす」の各ブロックは必要ありません。消去します。不要な変数も削除します。



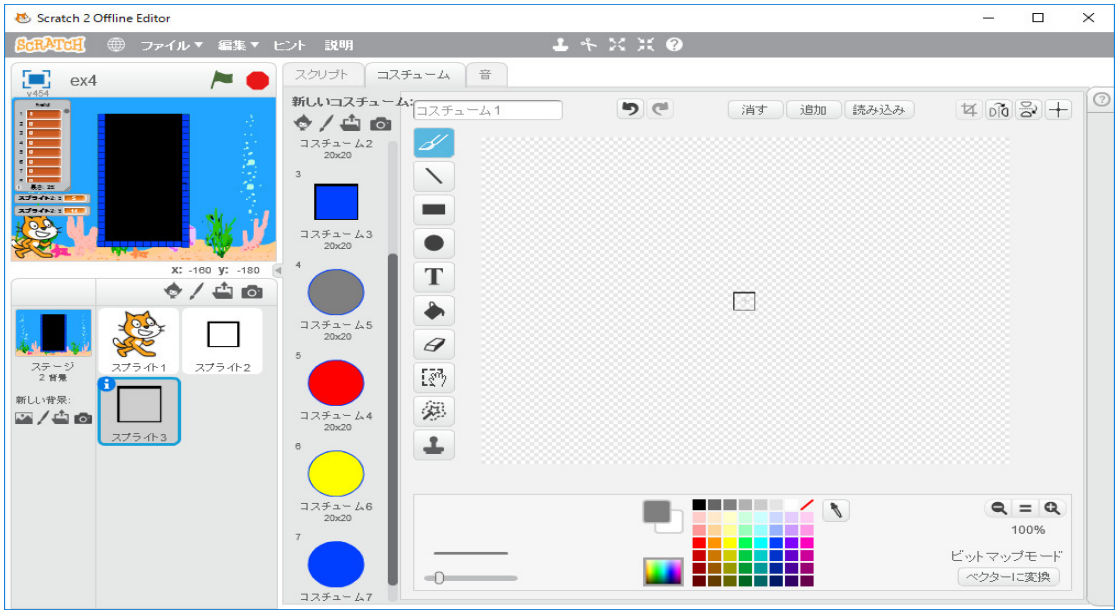
このプログラムを参考に、必要なフィールドの大きさ、位置、一つの「ぷよ」のサイズを決めます。「フィールドは縦 12 マス×横 6 マスの格子で構成される」とあるので、スプライトをもっと大きくできます。20 ドット× 20 ドットにします。まず必要なスプライトを作りましょう。新しいスプライトを「新しいスプライトを描く」で作ります。



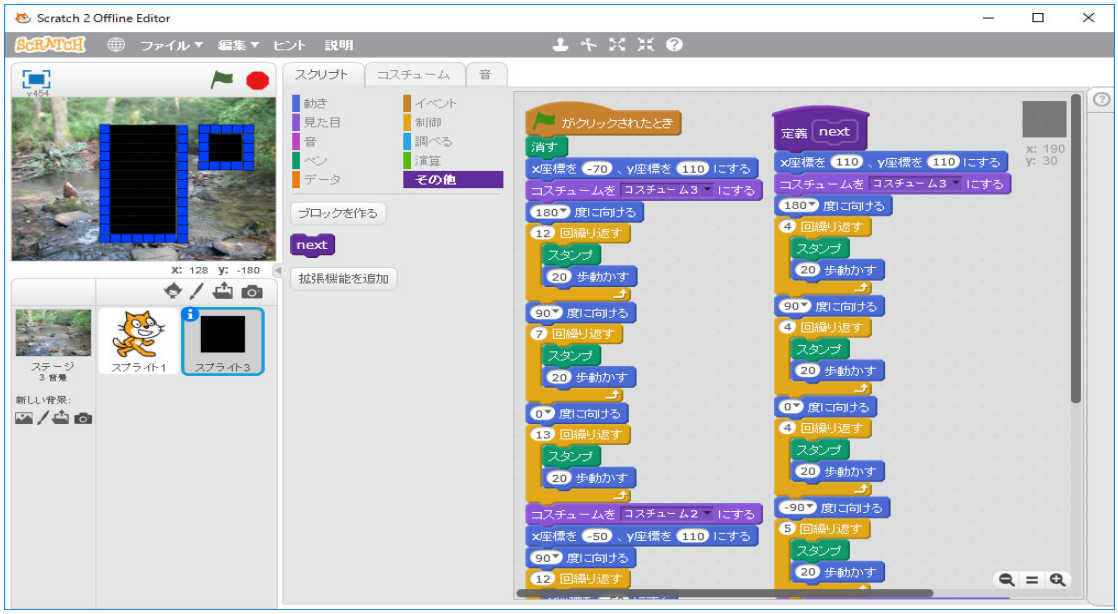
まずは 20 ドット× 20 ドットの正方形を作ります。作り方はテトリスの説明でやりましたから、忘れた方はそこを参照してください。



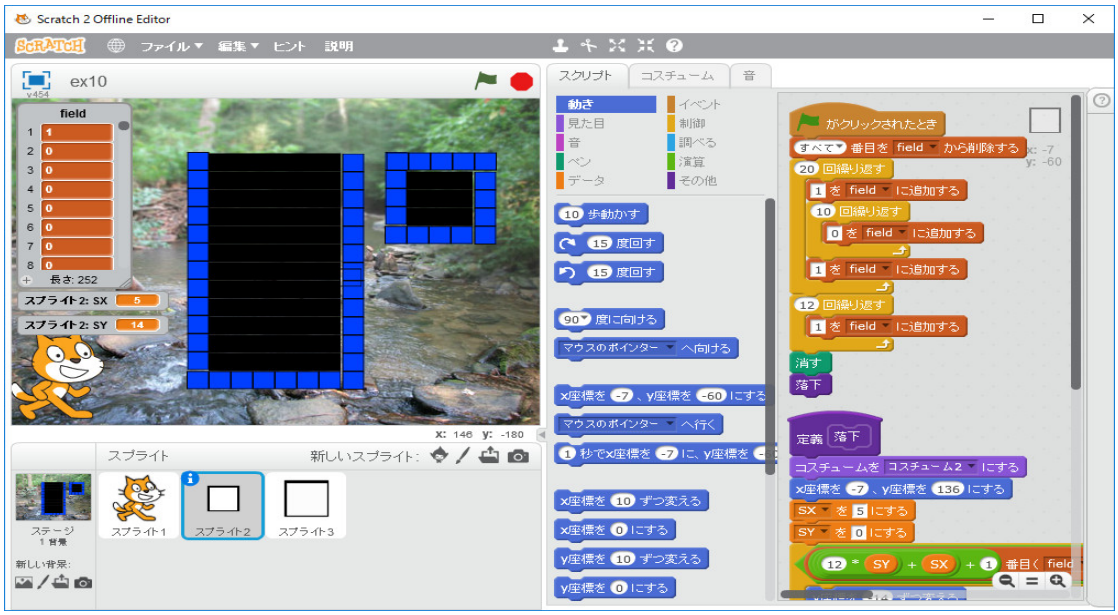
コスチュームを二つコピーし、黒と青の正方形にします。同じく、「ぷよ」として使う 20 ドット×20 ドットの 4 色の○を作ります。



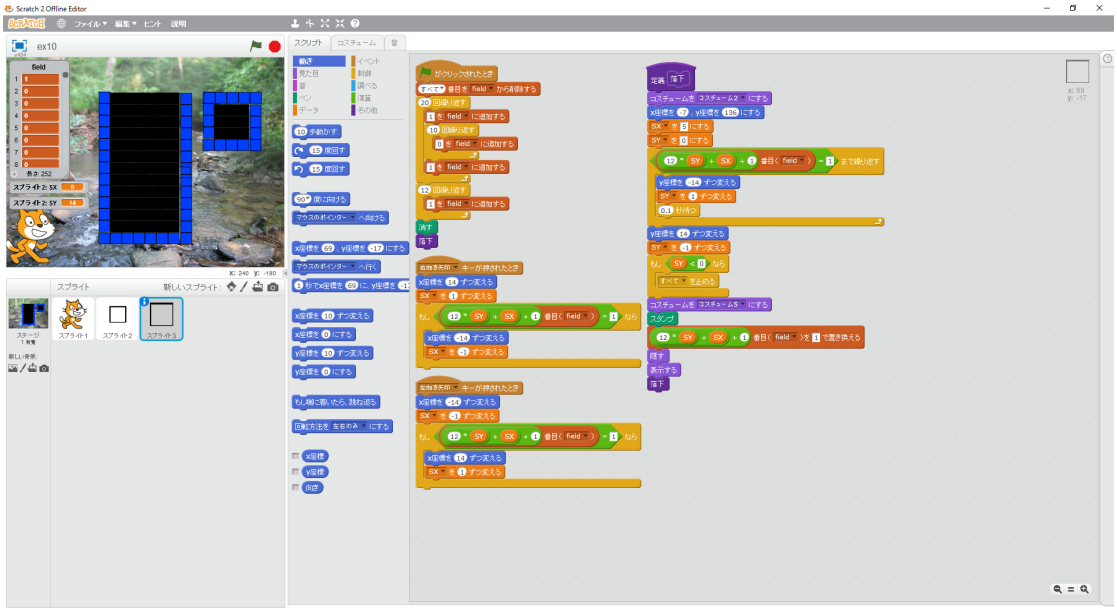
これでスプライトは完成です。贅沢は言いません。「スプライト 3」を右クリックし、「ローカルファイルに保存」を選択し、保存します。このプログラムを名前を付けて保存しておきます。次に背景を作ります。新しい Scratch を立ち上げます。新しいスプライトの「ファイルから新しいスプライトをアップロード」をクリックし、先ほど保存したスプライトを読み込みます。背景のための画像を Scratch で描きます。



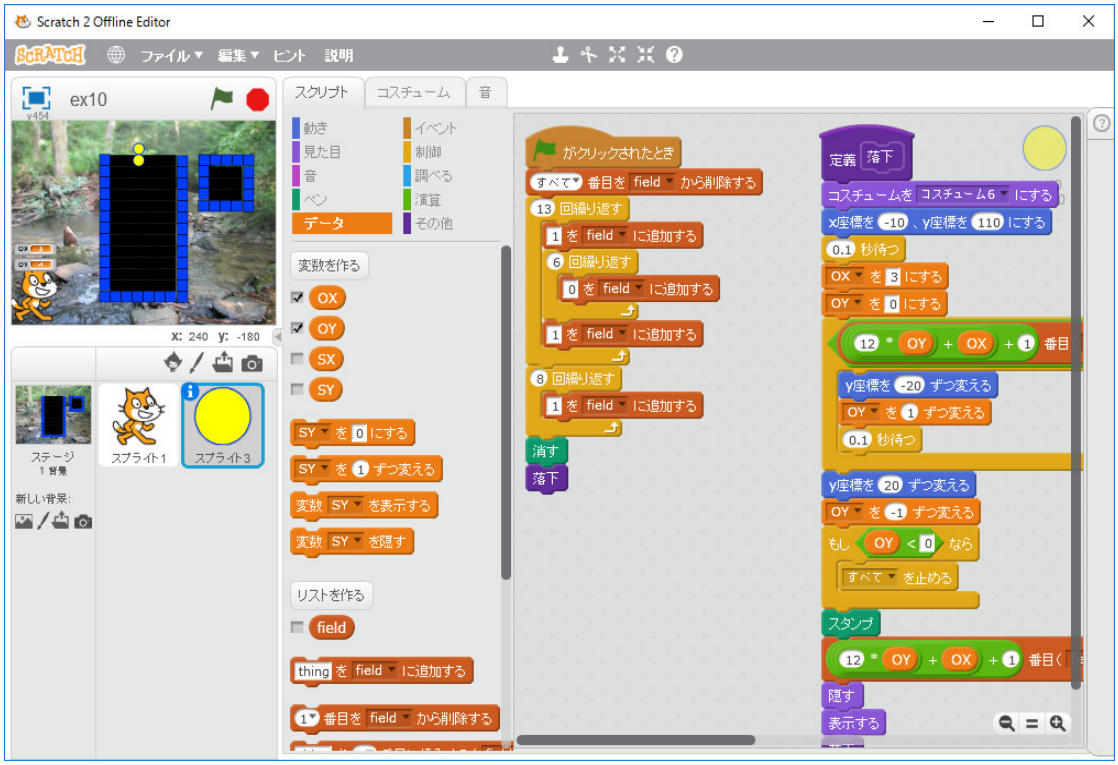
画面を右クリックして、保存します。背景を作る前に使っていたプログラムを立ち上げます。今作った背景を読み込みます。必要なくなった背景は削除しておきます。



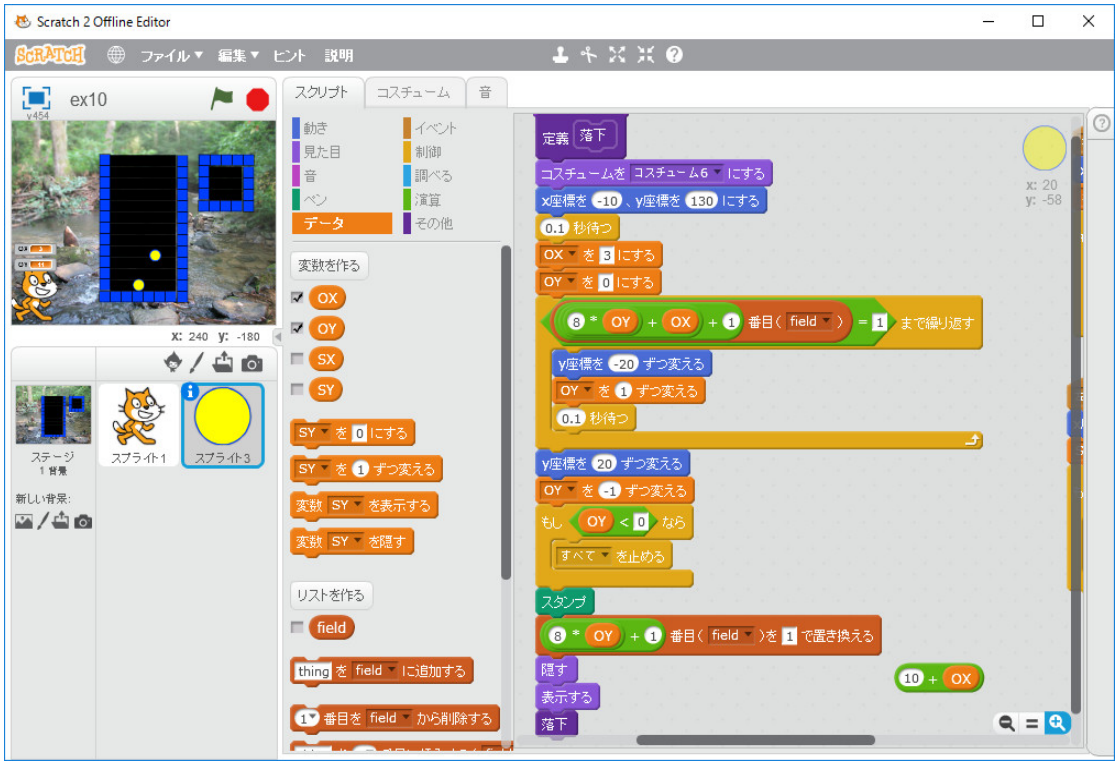
テトリス用のスプライトのスクリプトを表示し、各ブロックをぷよぷよ用のスプライトにドラッグしてコピーします。



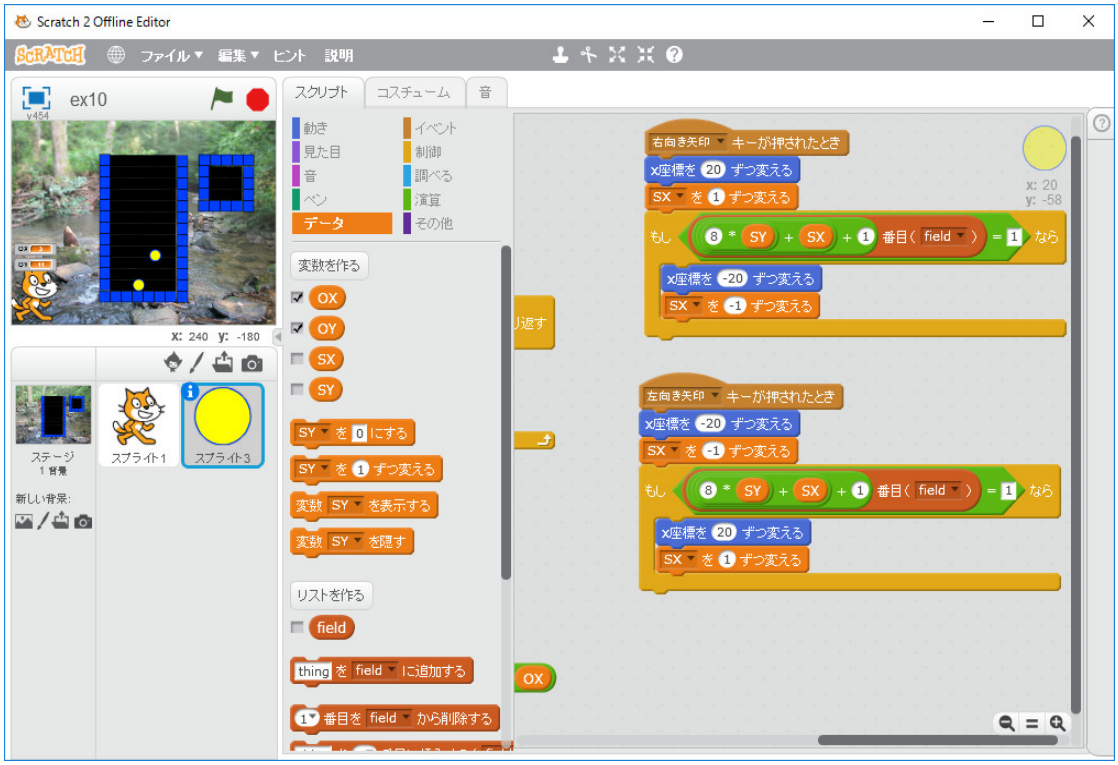
テトリス用のスプライトは必要なくなったので、削除します。プログラムを修正していきます。フィールドは枠外の最上段を加えて、14 段× 8 列として、リスト field に初期値を下図のように設定します。



ブロック「落下」を背景に合わせて、下図のように修正します。

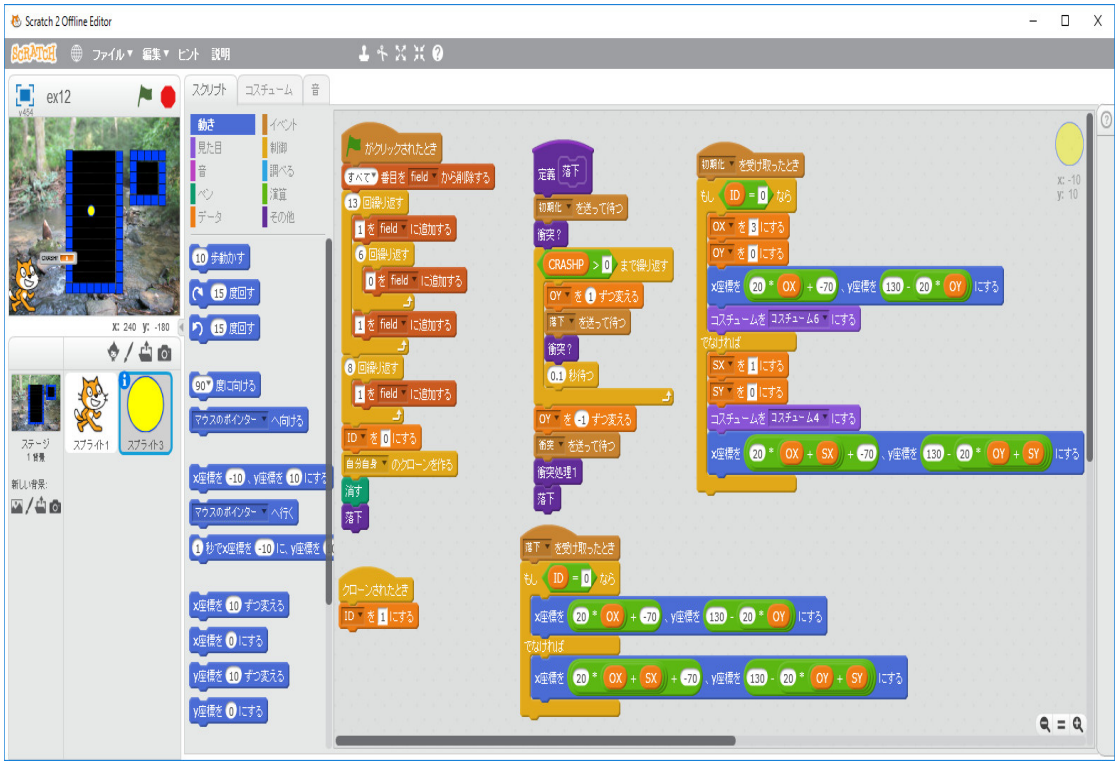


後のことを考え、新しいグローバル変数（すべてのスプライト用） OX, OY を作成し、SX,SY を OX,OY と、14 を 20 と、12 を 8 と入れ替え、OX=3, OY=0 と初期設定しています。次に、「右向き矢印キーが押されたとき」と「左向き矢印キーが押されたとき」を修正します。SX,SY を OX,OY と、14 を 20 と、12 を 8 と入れ替えるだけです。



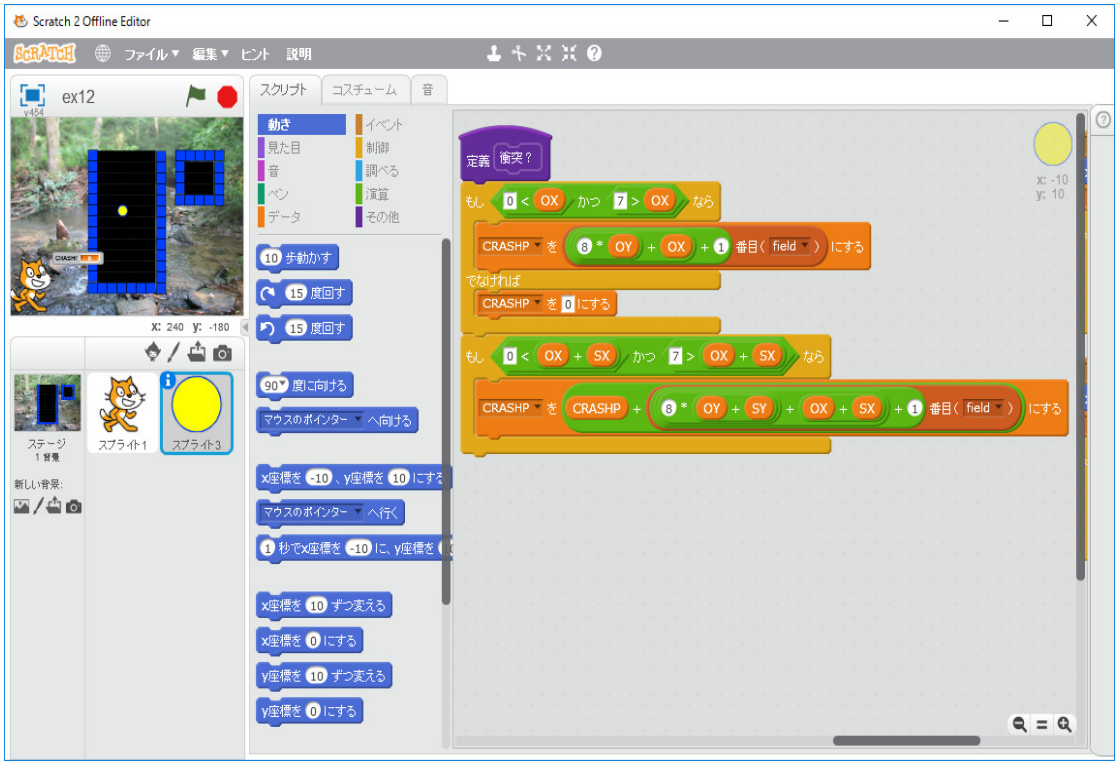
変な使い方をしなければ、正しく動いているように思います。ここまではテトリスと基本的に同じ考えでいいです。次は「ぷよぷよ」を落下させることを考えます。ここでも新たなスプライトをコピーして作らず、クローンで対応することにします。最初から完成版を作ろうとすると混乱しますから、段階を追って作っていきます。オリジナルのスプライトは黄色い球、クローンは赤い球に固定して、クローンを右に配置して、落下させます。ローカル変数（このスプライトのみ） ID を作ります。「旗がクリックされたとき」の field の設定後、自分自身のクローンを作るを挿入します。「クローンされたとき」で初期設定します。SX,SY はテトリスでは、クローンの相対座標を保持するためにローカル変数（このスプライトのみ）にしていて、衝突判定に苦労しましたが、ぷよぷよではクローンは一つなので、グローバル変数（すべてのスプライト用）で使います。一旦削除して、作り直します。こうすればオリジナルのスプライトで、クローンのフィールド座標での位置を知ることが出来ます。一般のプログラミング言語のようにプログラムリストの断片を印刷して眺めることが出来ないので、随分使い勝手が違っていて、Scratch で一か月近くプログラミングしていますがまだ混乱しています。プログラムが理解できるように、プログラムの断片が一つの画面に入るようにこまめに整理しながらプログラミングしないと上手くいきません。テトリスの場合は、平山尚著「プログラムはこうして作られる　プログラマの頭の中をのぞいてみよう」秀和システムというお手本があったので、大して頭を使わずに、適当に作っていれば、プログラムが出来上がりましたが、今回はお手本がないので、考えを整理しながら慎重に作る必要があります。

　クローンを作ったので、それに応じてプログラムを修正します。この部分は「テトリス」のプログラムと基本的に同じです。

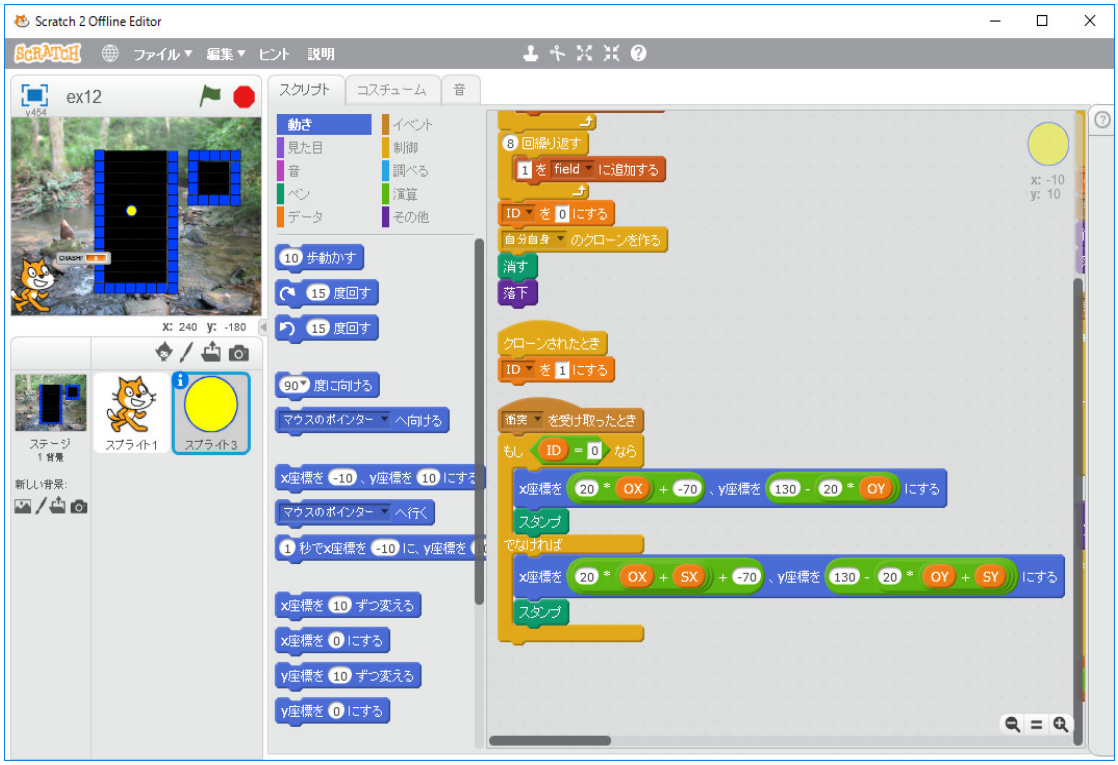


「旗がクリックされたとき」の処理は field の初期化と自身の ID=0 のセットとクローンの生成の後、「落下」に処理を渡します。「クローンされたとき」は単に自身の ID= 1 のセットだけをします。「落下」は終端再帰で無限ループになっています。「Game Over」で停止します。「落下」の先頭でオリジナルな「ぷよ」とクローンの「ぷよ」の初期化をします。何色の球にするかや初期位置

の設定に必要な処理をします。これは「初期化」というメセージを送ることで実行します。次にブロック「衝突？」で底や積みあがった「ぷよ」に衝突したかどうかを変数 CRASHP にセットします。

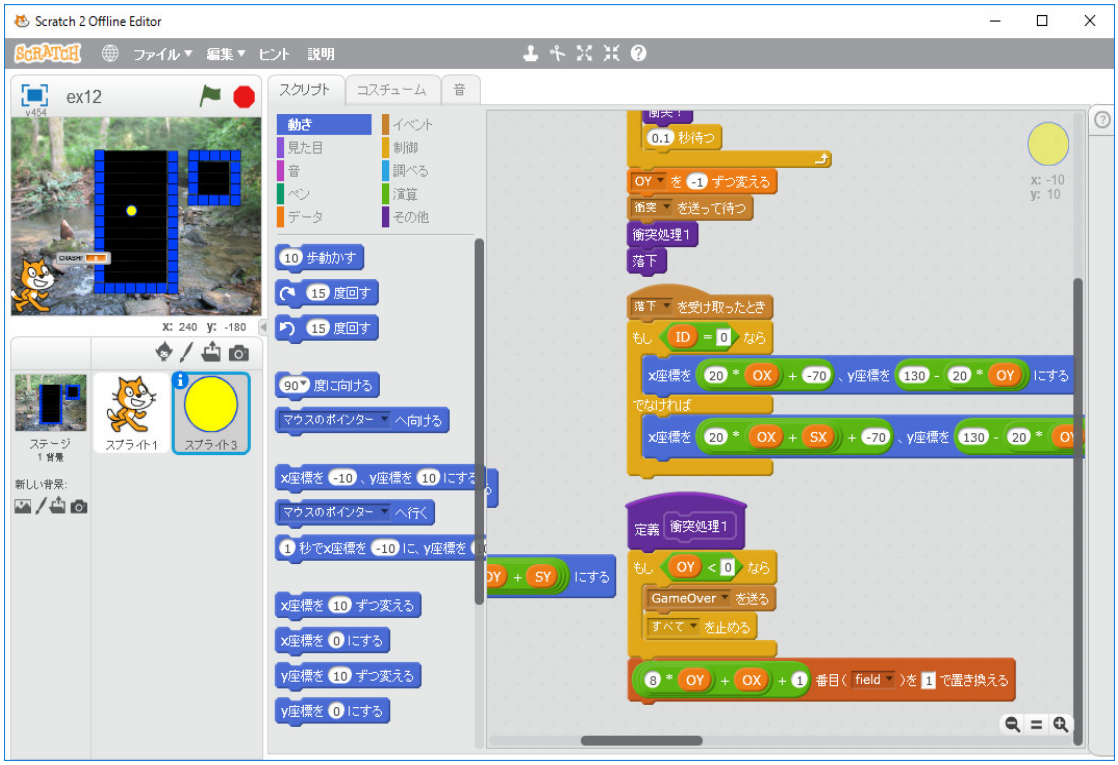


ブロック「衝突？」では、オリジナルのぷよとクローンのぷよのフィールド座標が壁以外の障害物にぶつかっているか（field で 1 となっているか？）を調べて、CRASHP をセットしている。SCRASHP が 0 の間は落下します。OY を一つ減らし「落下」のメセージを送って「ぷよぷよ」を表示させます。「0. 1 秒待つ」を挿入して、落下のスピードを調節しています。「$CRASHP > 0$」になったら、 衝突したということですから、OY を一つ減らし（上に移動し）、「衝突」のメセージを送り、現時点では仮にスタンプを押しています。

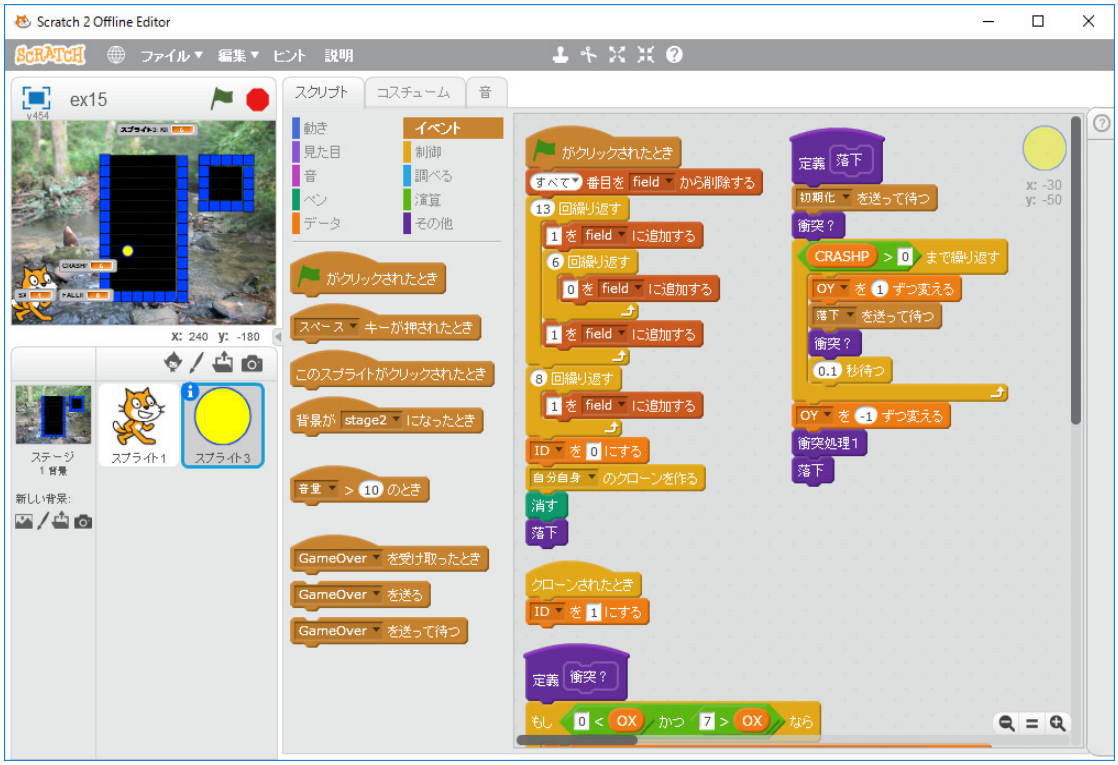


「衝突を受け取ったとき」は現時点では（後で修正しなければなりませんが)、現在のフィールド座標の位置にスタンプを押しています。

次に、ブロック「衝突処理 1 」で現時点では単に、オリジナルな「ぷよ」の位置を field にセットしています。

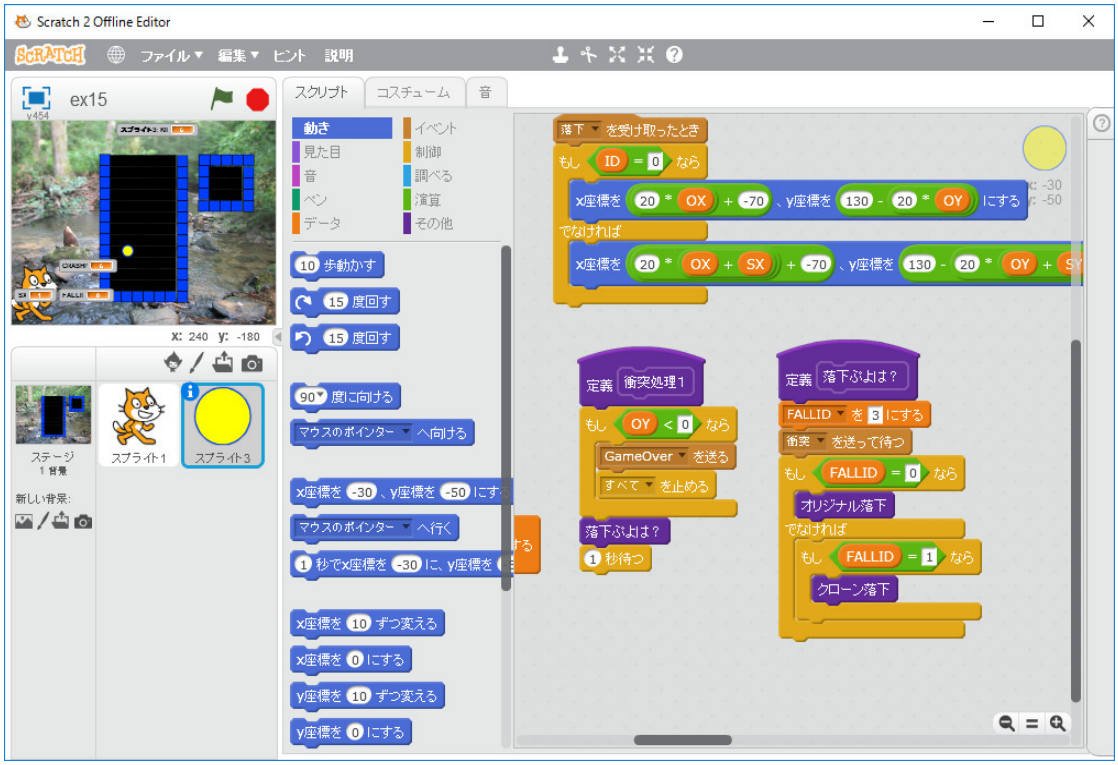


これらの処理は次の段階で修正します。最後に「落下」を再帰呼び出しし、無限ループになっています。

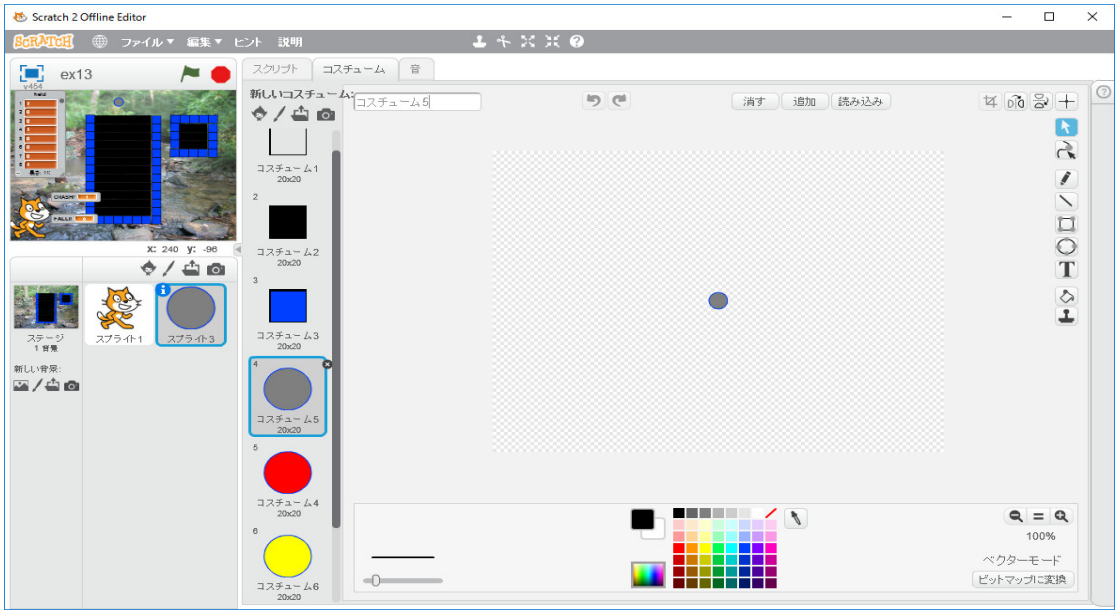


実際のゲームでは、下が空いている「ぷよ」の方は下まで落下する必要があります。その処理

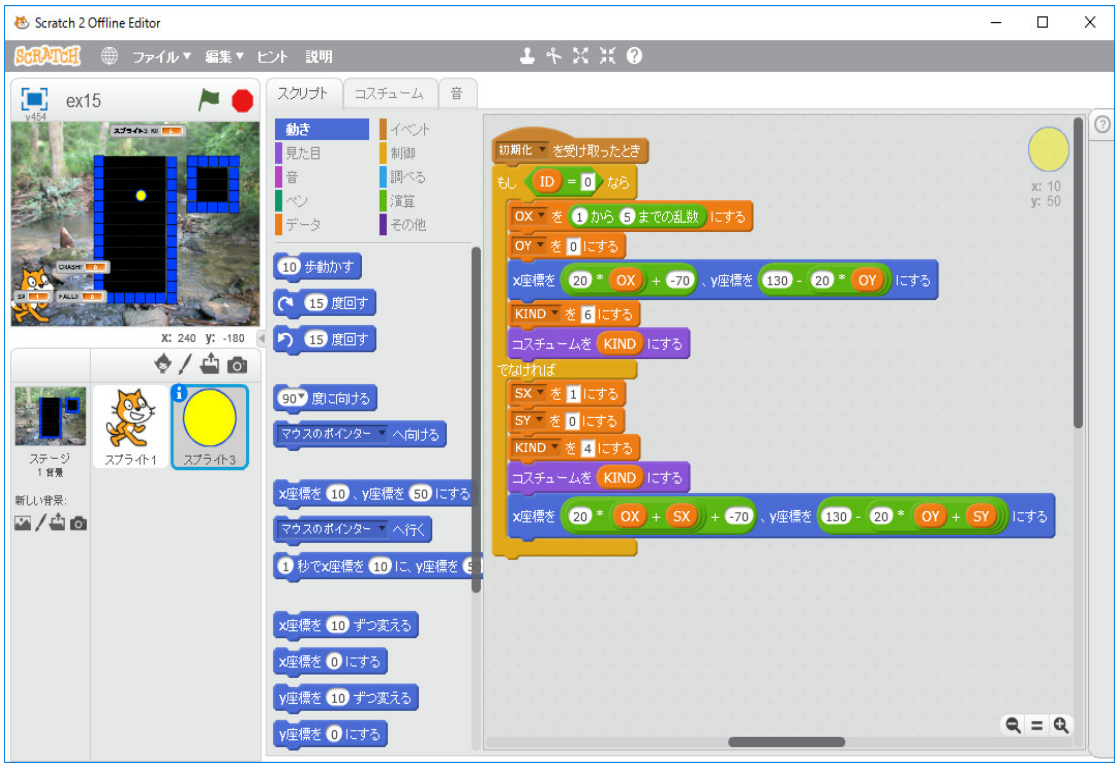
をプログラミングしましょう。これはメッセージ「衝突」とブロック「衝突処理１」が関係しています。どのように処理を割り振るべきか色々やり方がありますが、考えてもわからないので、メッセージを使うのは厄介なので、すべてブロック「衝突処理１」で処理します。まず、メッセージ「衝突」で下が空いてなければ、スタンプを押し、落ちなければいけない「ぷよ」がオリジナルの「ぷよ」なら 0、クローンの「ぷよ」なら 1、どちらも落ちなければ 3 を変数 FALLID にセットしします。この処理をブロック「落下ぷよは？」で実行します。まず変数 FALLID とブロック「落下ぷよは？」を作ります。



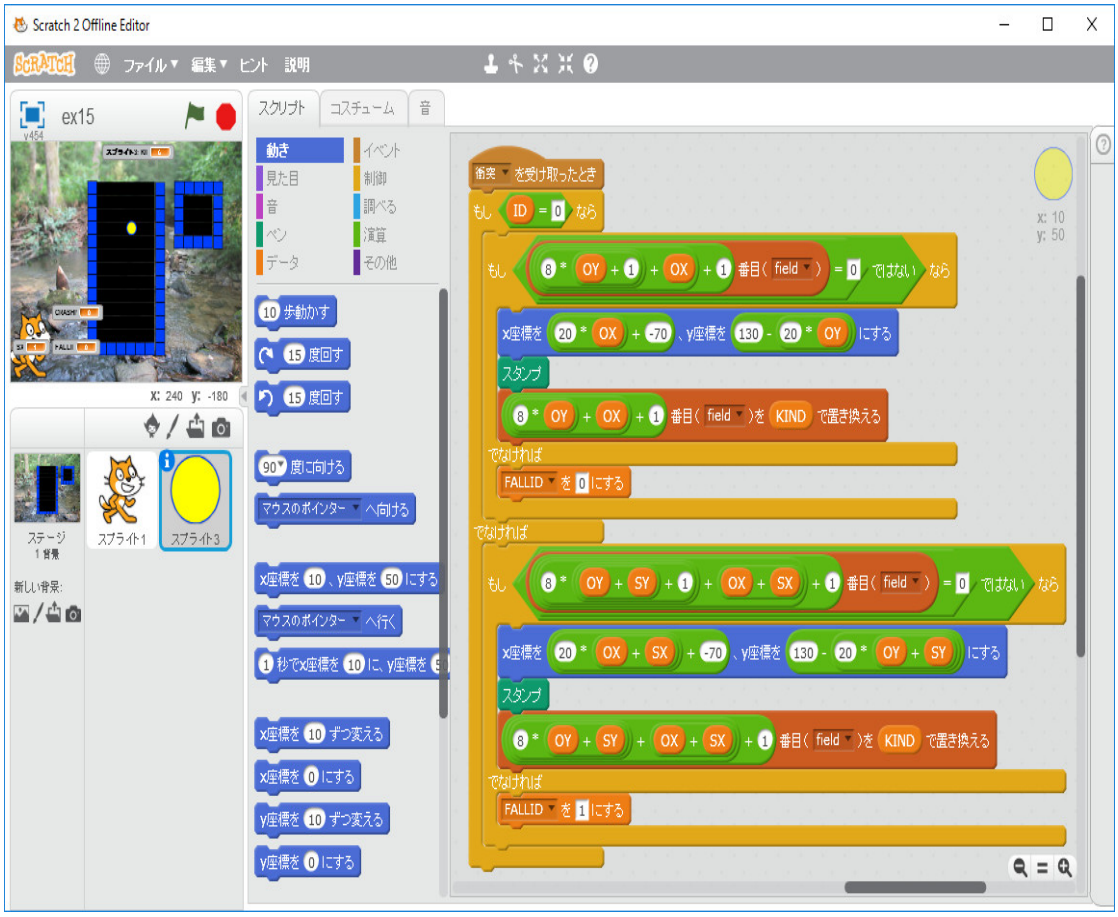
ここで大事なことに気が付きました。基本的なルールによると「ぷよ」が４個以上くっつくと消滅し得点となるとありますが、これを調べるためには、 field が 0 か 1 かでなく、どの色の球が積みあがっているか分からなければいけないので、上の図の処理で field に 1 をセットしているのでは駄目です。スプライトが今何の色の球なのかを保持している変数が必要です。この処理はオリジナルの「ぷよ」でやっているので、格好悪いですが、変数 KIND0 と KIND1 で保持するか、かっこよくローカル変数で保持するなら、メッセージ「衝突」で処理することになります。折角、Scratch では並列処理がブラックボックス化されてサポートされていますから、練習のため、後者の方法でやってみることに方針変更します。この為には、コスチュームを変数で変更できる必要があります。プログラムを一旦保存して、確かめてみましょう。ぷよのスプライトのコスチュームを開きます。



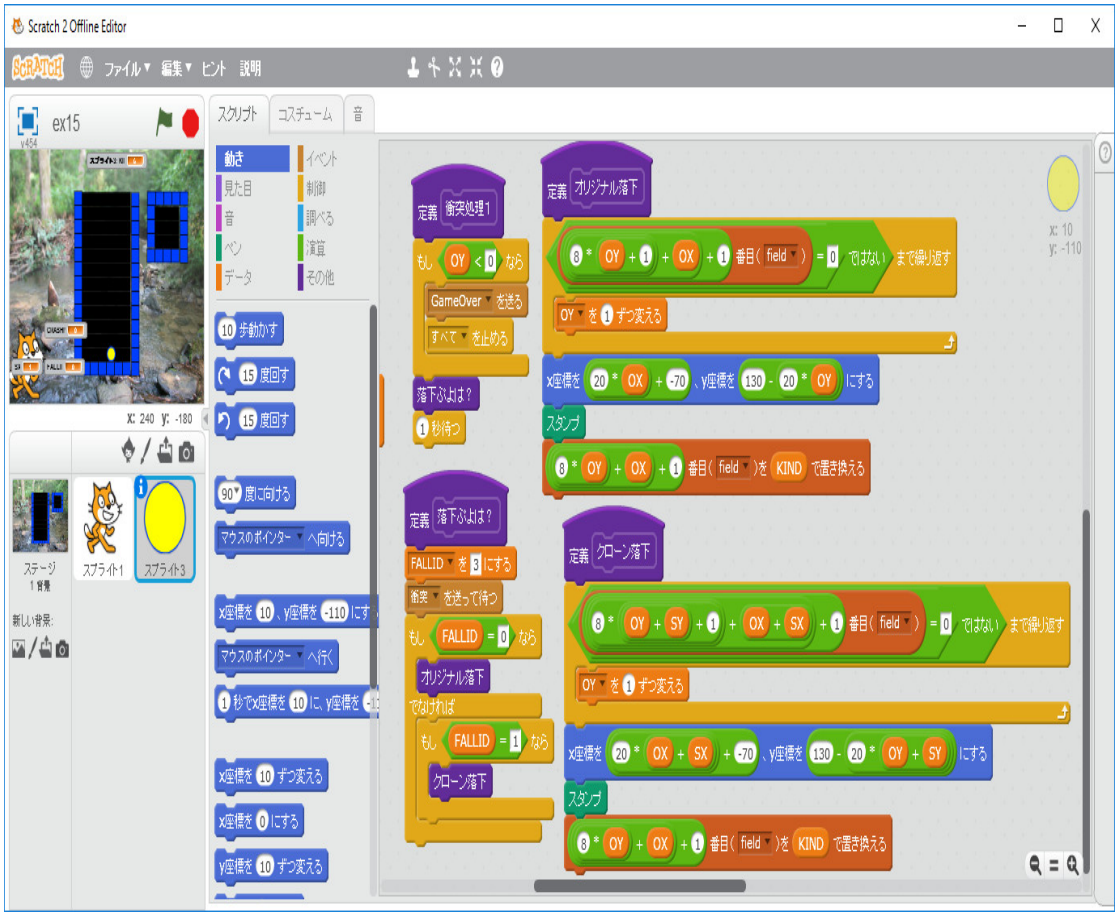
画像を作る画面の左上隅のエディターで球のコスチュームの名前をコスチュームを除いて数字だけにします。これで、KIND=4,5,6,7 とし、「コスチュームを KIND にする」でコスチュームを変えることが出来ることをチェックします。大丈夫です。このままプログラミングを続けます。ローカル変数（このスプライトのみ）　KIND を作ります。「初期化を受け取ったとき」を以下のように修正します。



ブロック「衝突処理１」、ブロック「落下ぷよは？」、及び「衝突を受け取ったとき」の定義を下図のようにします。どのように定義すべきかは明らかです。上の議論に合わせて修正しています。

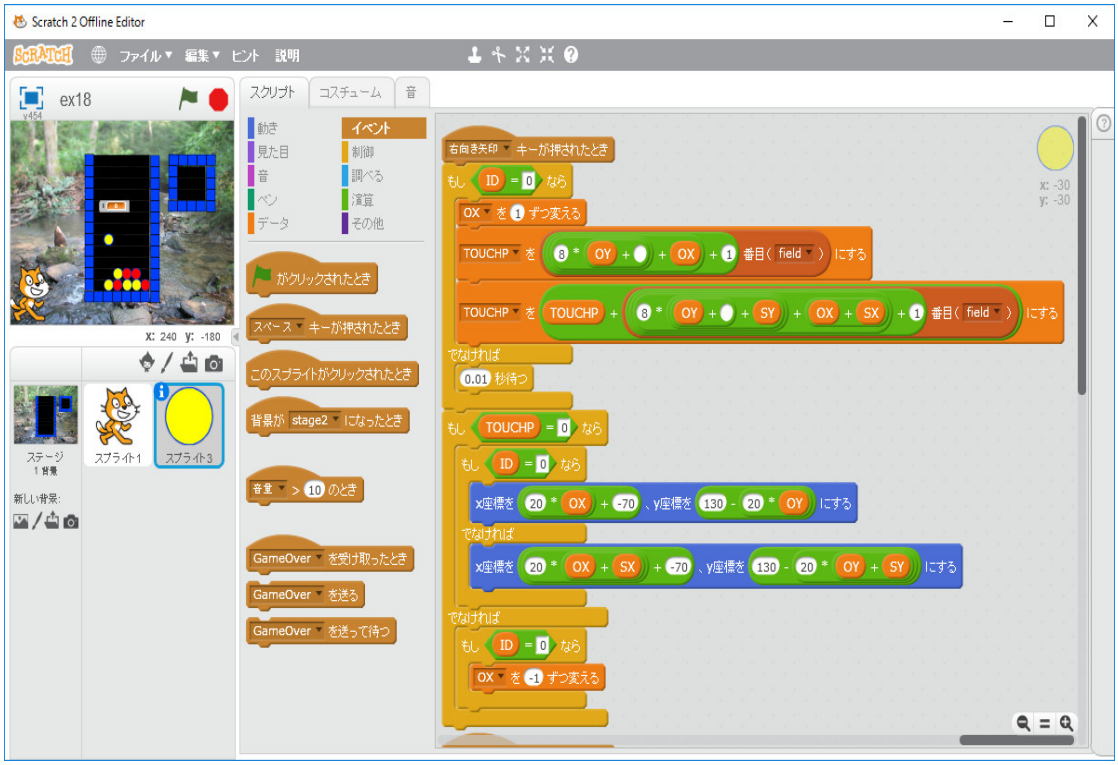


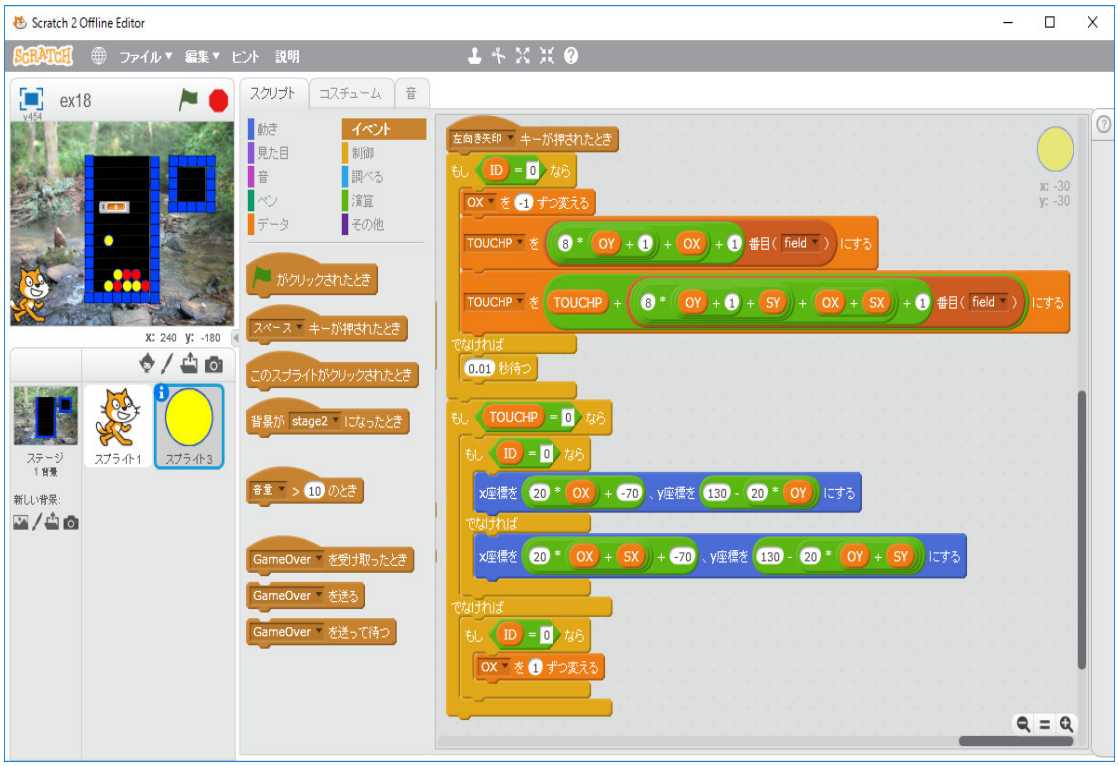
ブロック「落下ぷよは？」の続きを定義します。FALLID の値に応じて、すべきことを書きます。即ち、FALLID=0 ならブロック「オリジナル落下」をよび、FALLID=1 ならクローンの KIND はクローンしか知りませんから、メッセージ「クローン落下」送ります。



「初期化を受け取ったとき」で、OX の初期値を「1 から 5 までの乱数」として、実行してみます。矢印キーを押さなければ、正しいみたいです。ブロック「オリジナル落下」」を呼ぶ前やメッセージ「クローン落下」を送る前に OY の値を保存しておくべきだったかも分かりません。必要になったら考察しましょう。

次は、「右向き矢印キーが押されたとき」と「左向き矢印キーが押されたとき」のプログラムを修正します。これはテトリスで経験があるので簡単なはずです。変数 TOUCHP を作ります。





とすれば良いです。

次は最大の難所である「ぷよ」が４個以上つながっていれば消す作業をプログラミングします。これはここまでのように Scratch のスクリプトの画面を眺めて、頭の中でアルゴリズムを組み立

て、ブロックを並べプログラミングしていくことは私にはできません。Scratch をいったん離れ、別の言語（C++ でも Java でも Python でも自分が得意な言語なら何でも良いですが）でアルゴリズムを書いてみてから、それを見ながら、Scratch のプログラムに変換するようにします。情報数学の授業では現在は Python を使っているので、Python で練習してみます。

最初に考えることは

```
import random

field = []
for k in range(13):
    field.append(1)
    for i in range(6):
        field.append(0)
    field.append(1)
for i in range(8):
    field.append(1)

for k in range(80):
    x = random.randint(1,6)
    y = random.randint(1,12)
    field[8*y+x] = random.randint(4,7)
```

のようなプログラムで、field に

```
[1, 0, 0, 0, 0, 0, 0, 1,
 1, 5, 7, 5, 4, 4, 6, 1,
 1, 0, 6, 0, 6, 0, 0, 1,
 1, 4, 5, 0, 0, 6, 5, 1,
 1, 7, 7, 4, 6, 4, 6, 1,
 1, 0, 0, 4, 0, 7, 4, 1,
 1, 0, 6, 6, 4, 7, 4, 1,
 1, 4, 6, 0, 6, 7, 4, 1,
 1, 4, 0, 6, 6, 0, 0, 1,
 1, 6, 0, 7, 6, 7, 4, 1,
 1, 6, 7, 0, 7, 0, 0, 1,
 1, 7, 4, 0, 0, 0, 6, 1,
 1, 4, 0, 5, 5, 6, 5, 1,
 1, 1, 1, 1, 1, 1, 1, 1]
```

のようにセットして、4 個以上連結している球があるかどうか調べる必要があるわけです。Scratch ではリストのリストも作れますが、中のリストは文字列の並びに変換され、文字列のリストとなるので、通常のリストのリストとしては使えません。Scratch では一次元のリストしか使えないと思った方がいいです。ここでは単純に、 field 全体を探索して、4 個以上連結している球があれば、それらの座標の組をゲットするプログラムを作ります。Python はリストのリストが作れるので、

４個以上連結している球の組すべてを一気に取り出すことが出来ます。まずは Scratch のことは考えず、Python だけに集中して、プログラミングしてみます。

効率よくチェックするためには天井も 1 で埋めておく必要があることに気づきます。毎回同じ乱数を生成するようにしておきます。

```
import random
random.seed(0)

def clearlist(l):
    for i in range(8):
        l.append(1)
    for k in range(13):
        l.append(1)
        for i in range(6):
            l.append(0)
        l.append(1)
    for i in range(8):
        l.append(1)

field = []
clearlist(field)

for k in range(80):
    x = random.randint(1,6)
    y = random.randint(1,13)
    field[8*y+x] = random.randint(4,7)
```

のようなプログラムで、field に

```
[1, 1, 1, 1, 1, 1, 1, 1,
 1, 4, 4, 0, 0, 4, 7, 1,
 1, 5, 0, 4, 6, 7, 6, 1,
 1, 0, 4, 5, 4, 6, 4, 1,
 1, 4, 4, 0, 4, 6, 4, 1,
 1, 7, 6, 0, 0, 5, 7, 1,
 1, 4, 6, 0, 5, 0, 5, 1,
 1, 0, 0, 0, 0, 0, 4, 1,
 1, 0, 0, 0, 0, 4, 5, 1,
 1, 0, 0, 6, 0, 6, 6, 1,
 1, 7, 0, 6, 6, 4, 0, 1,
 1, 0, 7, 0, 7, 4, 5, 1,
 1, 0, 6, 4, 4, 4, 0, 1,
 1, 5, 7, 0, 6, 5, 5, 1,
 1, 1, 1, 1, 1, 1, 1, 1]
```

のようにセットして、４個以上連結している球があるかどうか調べます。

```
[1, 1, 1, 1, 1, 1, 1, 1,
 1, 0, 0, 0, 0, 0, 0, 1,
 1, 0, 0, 0, 0, 0, 0, 1,
 1, 0, 0, 0, 0, 0, 0, 1,
 1, 0, 0, 0, 0, 0, 0, 1,
 1, 0, 0, 0, 0, 0, 0, 1,
 1, 0, 0, 0, 0, 0, 0, 1,
 1, 0, 0, 0, 0, 0, 0, 1,
 1, 0, 0, 0, 0, 0, 0, 1,
 1, 0, 0, 0, 0, 0, 0, 1,
 1, 0, 0, 0, 0, 0, 0, 1,
 1, 0, 0, 0, 0, 0, 0, 1,
 1, 0, 0, 0, 0, 0, 0, 1,
 1, 1, 1, 1, 1, 1, 1, 1]
```

のようなリスト ckd を準備し、探索が済んだ場所に 1 を配置していきます。

```
import random
random.seed(1)

def clearlist(l):
    for i in range(8):
        l.append(1)
    for k in range(13):
        l.append(1)
        for i in range(6):
            l.append(0)
        l.append(1)
    for i in range(8):
        l.append(1)

field = []
clearlist(field)

for k in range(100):
    x = random.randint(1,6)
    y = random.randint(1,13)
    field[8*y+x] = random.randint(4,7)

print(field)
```

```
rt = []
gp = []
ckd = []

def subget4s(x,y,k):
    global field, rt, gp, ckd
    gp.append([x,y])
    ckd[8*y+x]=1
    if (ckd[8*y+x+1]==0) and (field[8*y+x+1]==k):
        subget4s(x+1,y,k)
    if (ckd[8*y+x-1]==0) and (field[8*y+x-1]==k):
        subget4s(x-1,y,k)
    if (ckd[8*(y+1)+x]==0) and (field[8*(y+1)+x]==k):
        subget4s(x,y+1,k)
    if (ckd[8*(y-1)+x]==0) and (field[8*(y-1)+x]==k):
        subget4s(x,y-1,k)

def get4s():
    global field, rt, gp, ckd
    clearlist(ckd)
    for y in range(1,14):
        for x in range(1,7):
            if (ckd[8*y+x]==1): continue
            if (field[8*y+x]==0): continue
            gp = []
            subget4s(x,y,field[8*y+x])
            print(x,y,field[8*y+x],len(gp))
            if len(gp) >= 4:
                rt.append(gp)

get4s()
print('rt=', rt)
```

というプログラムを実行し、出力を整理すると

```
[1, 1, 1, 1, 1, 1, 1, 1,
 1, 7, 4, 5, 7, 0, 4, 1,
 1, 0, 7, 4, 0, 6, 0, 1,
 1, 0, 5, 0, 0, 6, 5, 1,
 1, 7, 4, 5, 6, 6, 0, 1,
 1, 5, 5, 5, 4, 0, 6, 1,
 1, 7, 5, 6, 0, 7, 0, 1,
 1, 7, 0, 5, 0, 4, 0, 1,
 1, 4, 0, 4, 4, 0, 6, 1,
```

```
 1, 6, 4, 7, 6, 0, 7, 1,
 1, 0, 4, 6, 6, 5, 7, 1,
 1, 7, 5, 4, 6, 5, 7, 1,
 1, 6, 0, 0, 7, 0, 7, 1,
 1, 6, 4, 0, 5, 7, 5, 1,
 1, 1, 1, 1, 1, 1, 1, 1]

1 1 7 1
2 1 4 1
3 1 5 1
4 1 7 1
6 1 4 1
2 2 7 1
3 2 4 1
5 2 6 4
2 3 5 1
6 3 5 1
1 4 7 1
2 4 4 1
3 4 5 5
4 5 4 1
6 5 6 1
1 6 7 2
3 6 6 1
5 6 7 1
3 7 5 1
5 7 4 1
1 8 4 1
3 8 4 2
6 8 6 1
1 9 6 1
2 9 4 2
3 9 7 1
4 9 6 4
6 9 7 4
5 10 5 2
1 11 7 1
2 11 5 1
3 11 4 1
1 12 6 2
4 12 7 1
2 13 4 1
4 13 5 1
```

```
5 13 7 1
6 13 5 1
rt= [
      [[5, 2], [5, 3], [5, 4], [4, 4]],
       [[3, 4], [3, 5], [2, 5], [1, 5], [2, 6]],
       [[4, 9], [4, 10], [3, 10], [4, 11]],
       [[6, 9], [6, 10], [6, 11], [6, 12]]
     ]
```

となります。Python は　Python 3.5.2 を使っています。

```
[1, 1, 1, 1, 1, 1, 1, 1,
 1, 7, 4, 5, 7, 0, 4, 1,
 1, 0, 7, 4, 0, 6, 0, 1,
 1, 0, 5, 0, 0, 6, 5, 1,
 1, 7, 4, 5, 6, 6, 0, 1,
 1, 5, 5, 5, 4, 0, 6, 1,
 1, 7, 5, 6, 0, 7, 0, 1,
 1, 7, 0, 5, 0, 4, 0, 1,
 1, 4, 0, 4, 4, 0, 6, 1,
 1, 6, 4, 7, 6, 0, 7, 1,
 1, 0, 4, 6, 6, 5, 7, 1,
 1, 7, 5, 4, 6, 5, 7, 1,
 1, 6, 0, 0, 7, 0, 7, 1,
 1, 6, 4, 0, 5, 7, 5, 1,
 1, 1, 1, 1, 1, 1, 1, 1]
rt= [
      [[5, 2], [5, 3], [5, 4], [4, 4]],
       [[3, 4], [3, 5], [2, 5], [1, 5], [2, 6]],
       [[4, 9], [4, 10], [3, 10], [4, 11]],
       [[6, 9], [6, 10], [6, 11], [6, 12]]
     ]
```

なので、プログラムはこれで良いみたいです。これは再帰探索のアルゴリズムで、プログラミングをしている私のようなアマチュアプログラマーの間でもよく知られたアルゴリズムです。このプログラムを Scratch のプログラムに変換します。Scratch は一次元のリストしか使えないので結果の表現が問題になります。

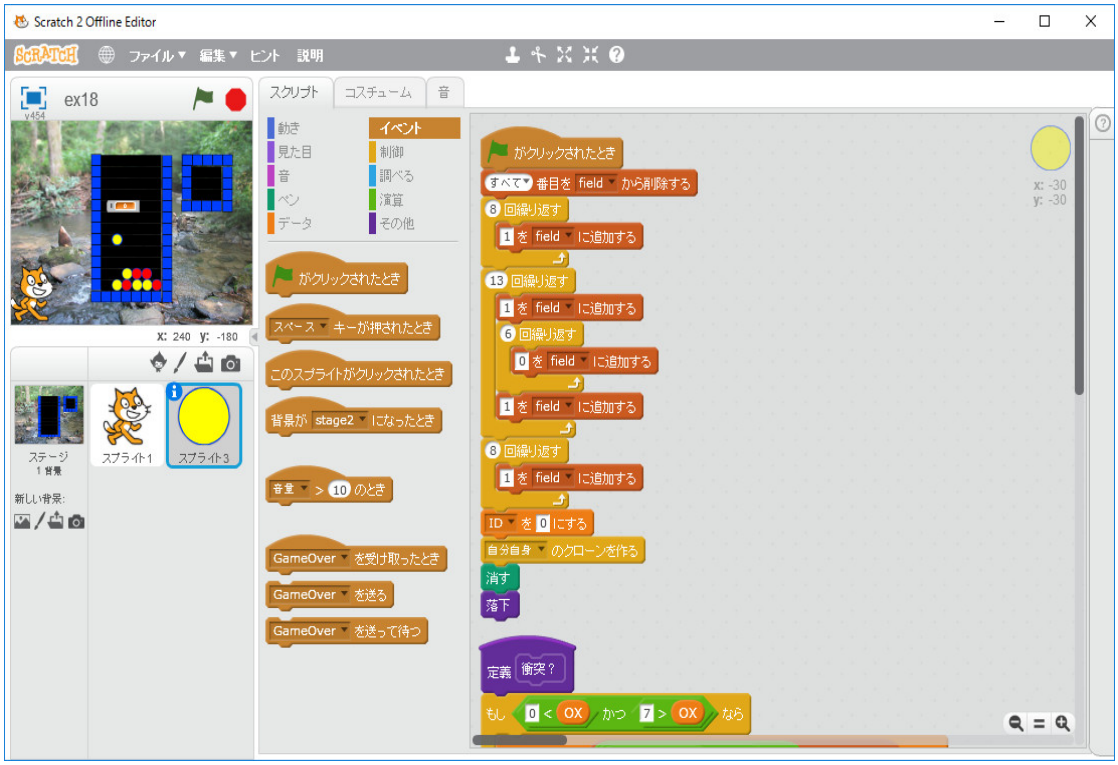
```
rt= [
      [[5, 2], [5, 3], [5, 4], [4, 4]],
       [[3, 4], [3, 5], [2, 5], [1, 5], [2, 6]],
       [[4, 9], [4, 10], [3, 10], [4, 11]],
       [[6, 9], [6, 10], [6, 11], [6, 12]]
     ]
```

を

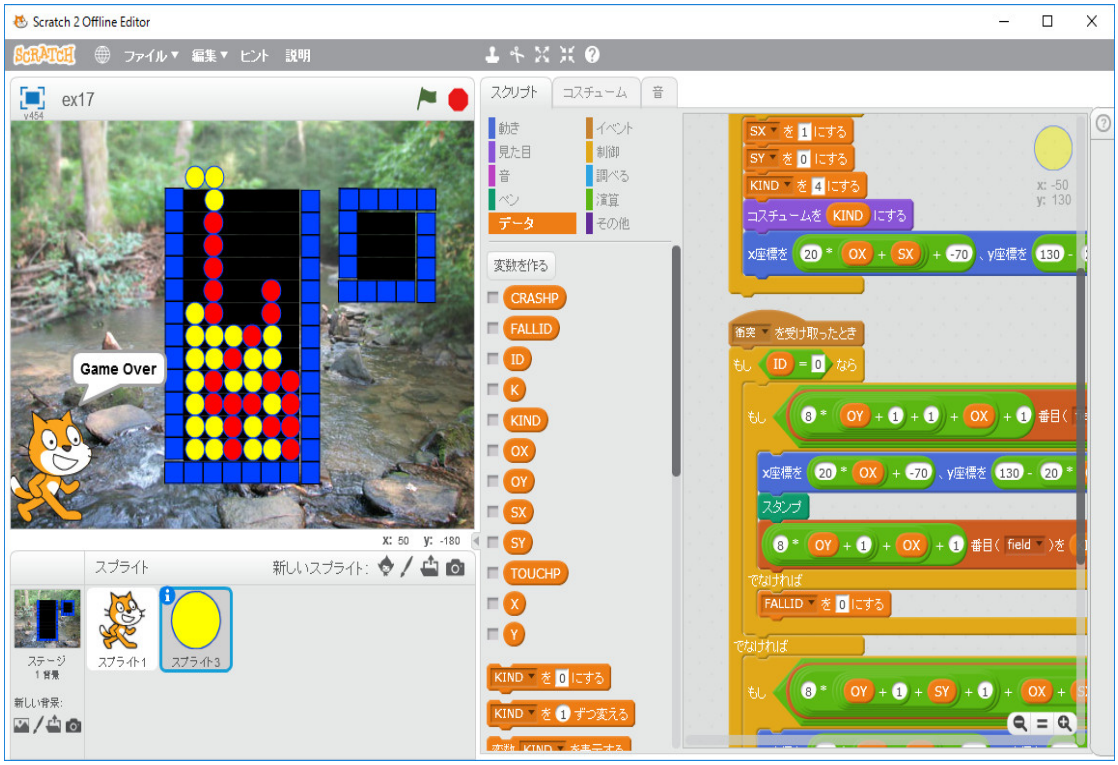
```
rt= [4, 5, 2, 5, 3, 5, 4, 4, 4,
       5, 3, 4, 3, 5, 2, 5, 1, 5, 2, 6,
       4, 4, 9, 4, 10, 3, 10, 4, 11,
       4, 6, 9, 6, 10, 6, 11, 6, 12
      ]
```

のように最初に連の長さをセットするようにします。これで準備が出来ました。 Scratch のプログラムに変換します。

まず field に天井 1 行増やし、関連するところを慎重に修正して、正しく動くようにします。まず、「旗がクリックされたとき」の定義を

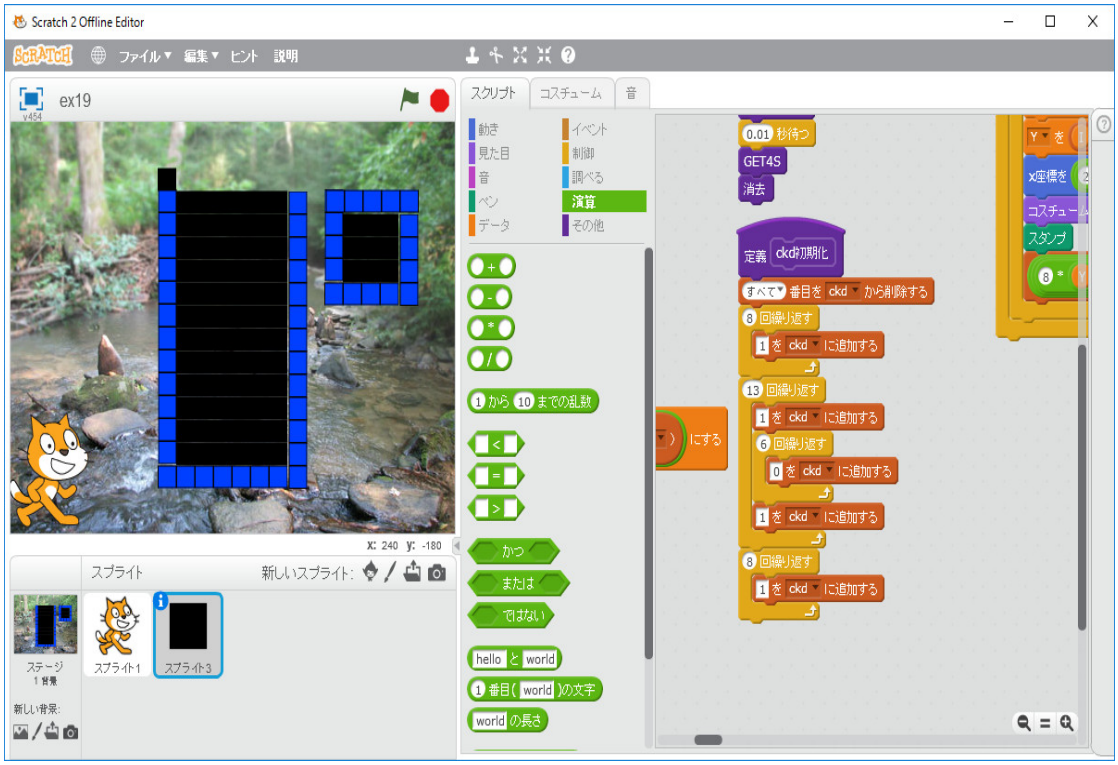


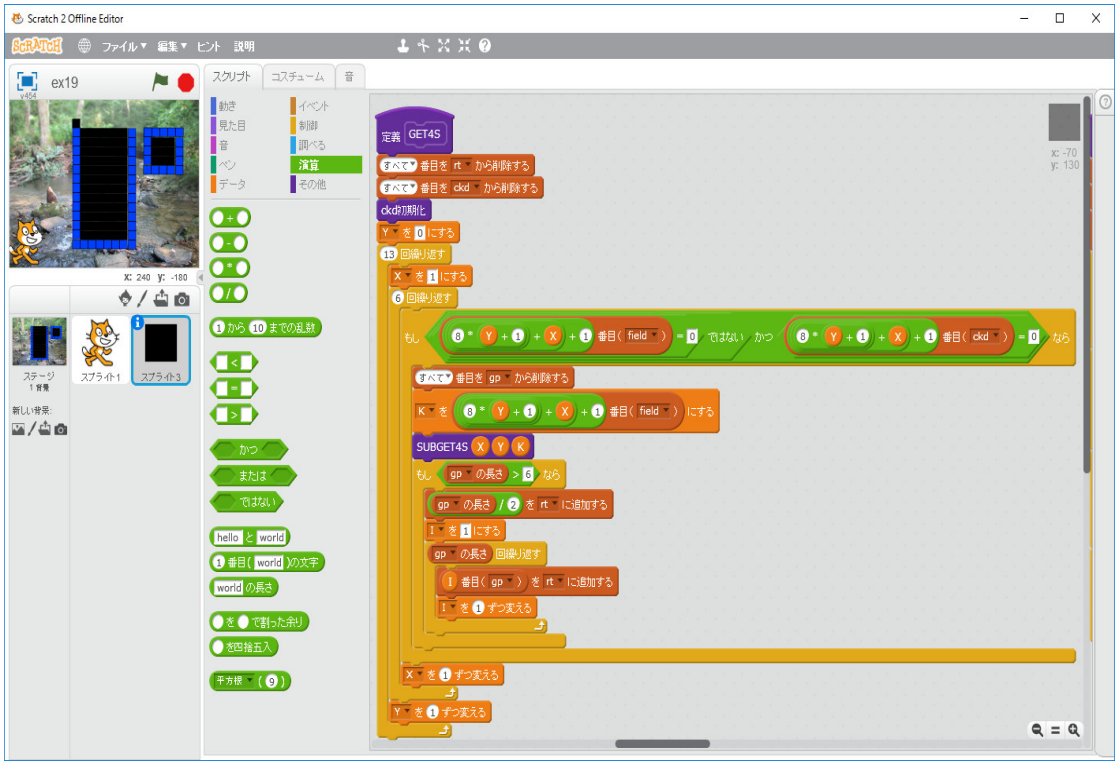
と直します。プログラムが平面的に並んでいるので、厄介なので、修正が最短限になるように修正するには、OX, OY に関するところは、画面の表示位置にも関連しているので触らず、field に関する部分を OY であったものを OY+1 に入れ替えます。沢山あるので一々図を表示しませんが、自分で修正してください。実行してみます。



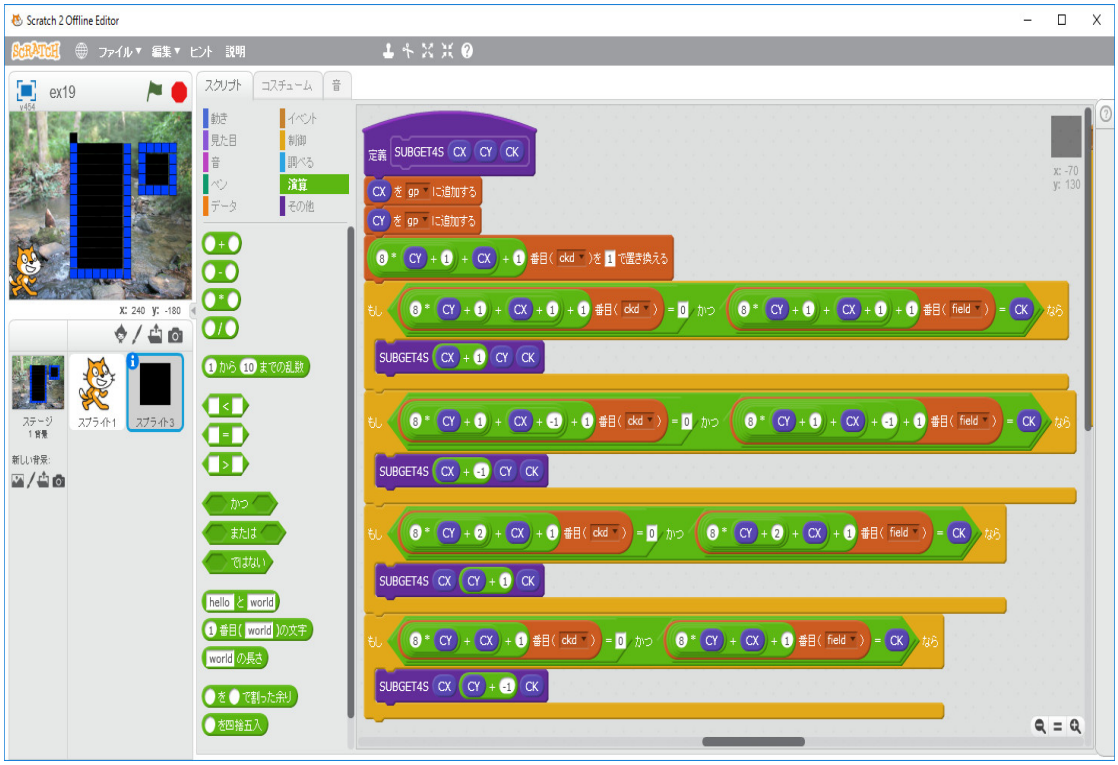
上手く修正できているみたいです。このような修正をするとバグが入り込み、厄介ですが、私のような素人は常に経験のないプログラムを作っているので、前もって必要なデータ構造を正確に予想できない（面倒なのでほとんど何も考えず、適当に作りながら考え、上手くいかなくなったら修正するの繰り返し）ので、このようなことになります。

まず、リスト ckd, gp, rt を作ります。変数 X, Y, K を作ります。ブロック「GET4S」と「SUB-GET4S CX, CY, CK」を作ります。ブロック「GET4S」は「衝突処理 1 」の定義の最後に追加します。更に、リスト ckd を初期化するプログラムを作ります。ブロック「GET4S」の定義は次の図のようになります。

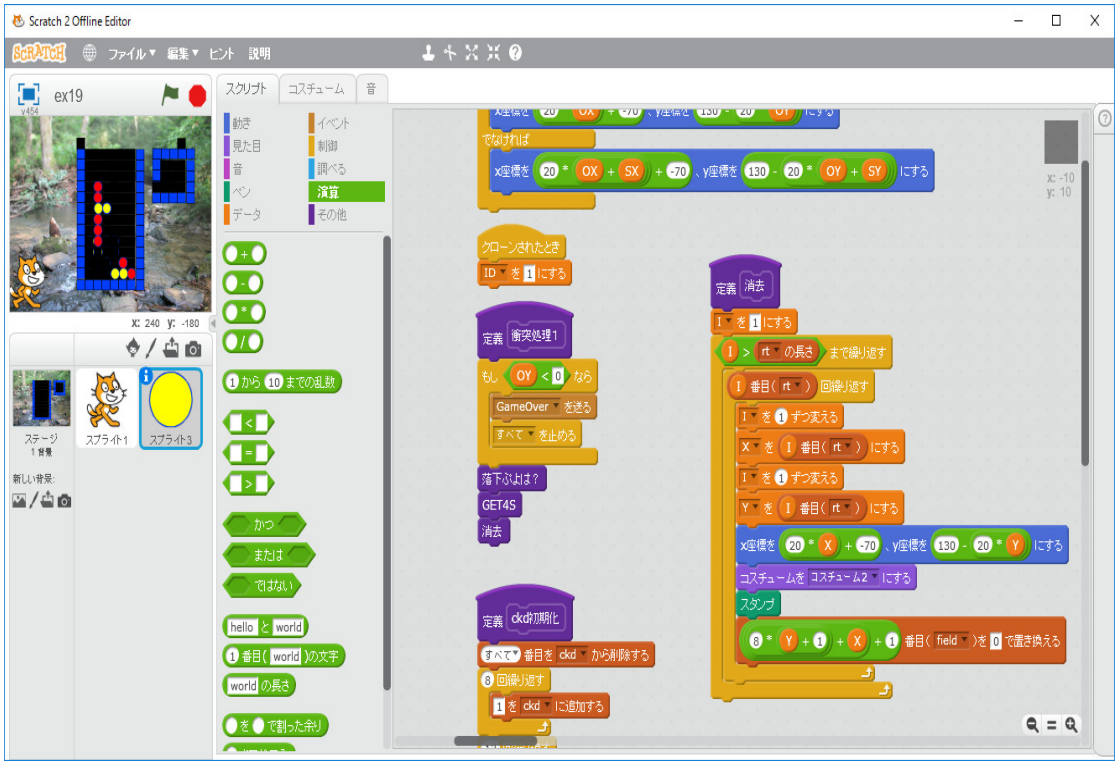




単に、Python のプログラムを直訳しているだけです。次に、ブロック「SUBGET4S CX, CY, CK」を定義します。

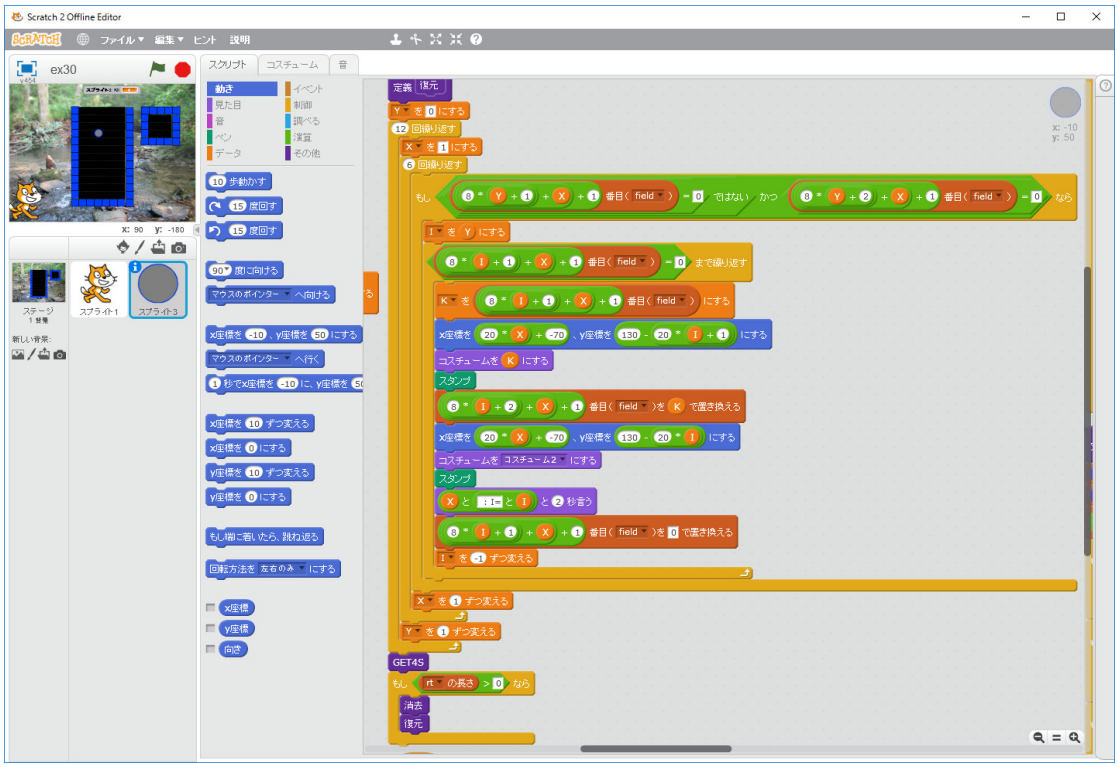


次に、４個以上くっついたぷよを消滅させるブロック「消去」を「衝突処理１」の定義の最後に追加します。ブロック「消去」の定義は下の図で定義されています。

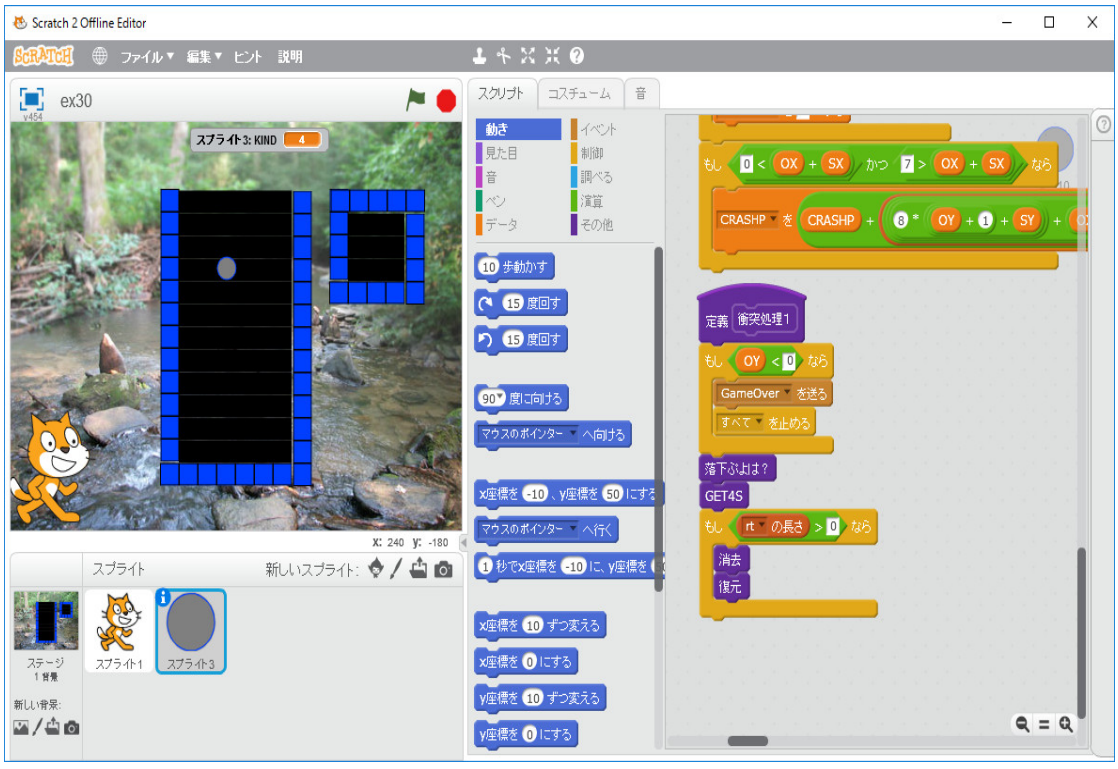


「ぷよ」の消滅により上にあった「ぷよ」が落下する処理をするブロック「復元」をブロック「消

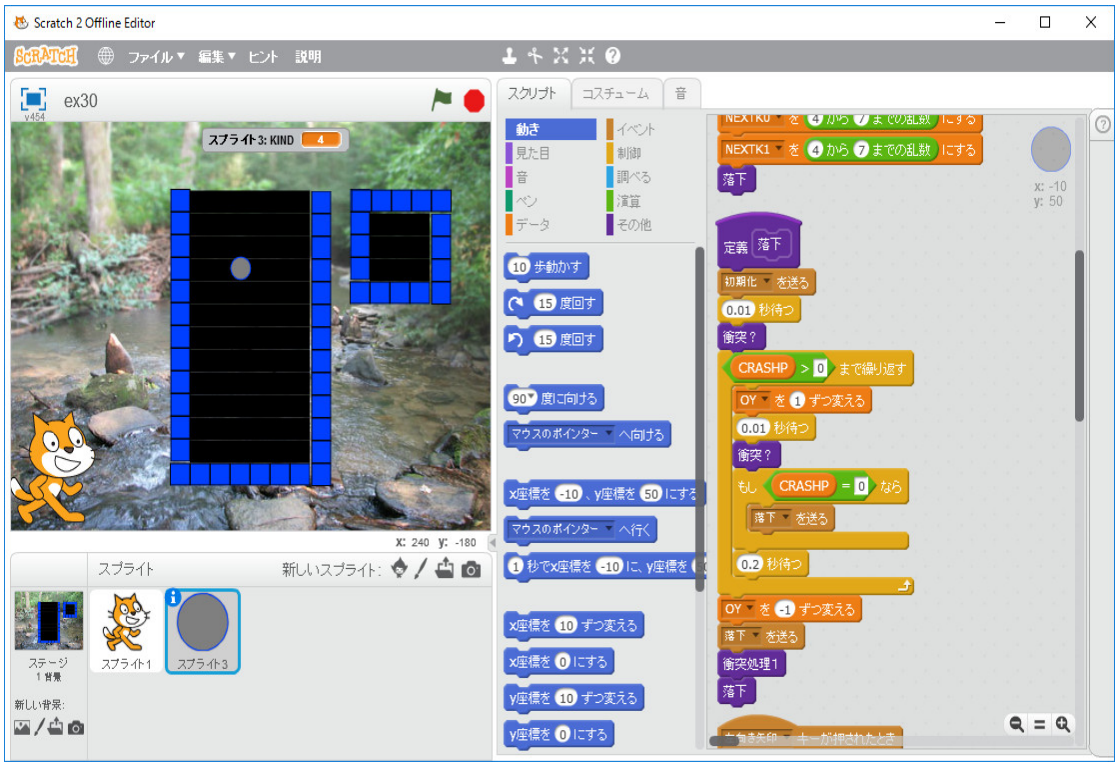
去」の後ろに追加する。



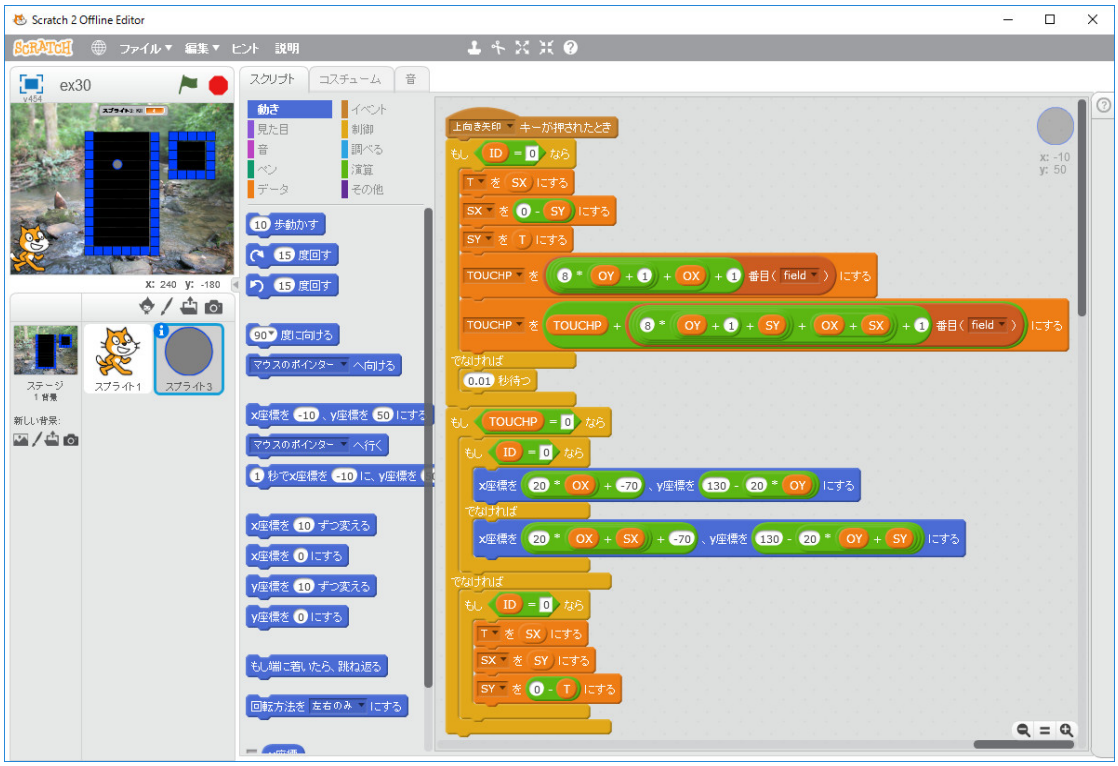
ブロック「衝突処理１」の定義を下図のように修正した。



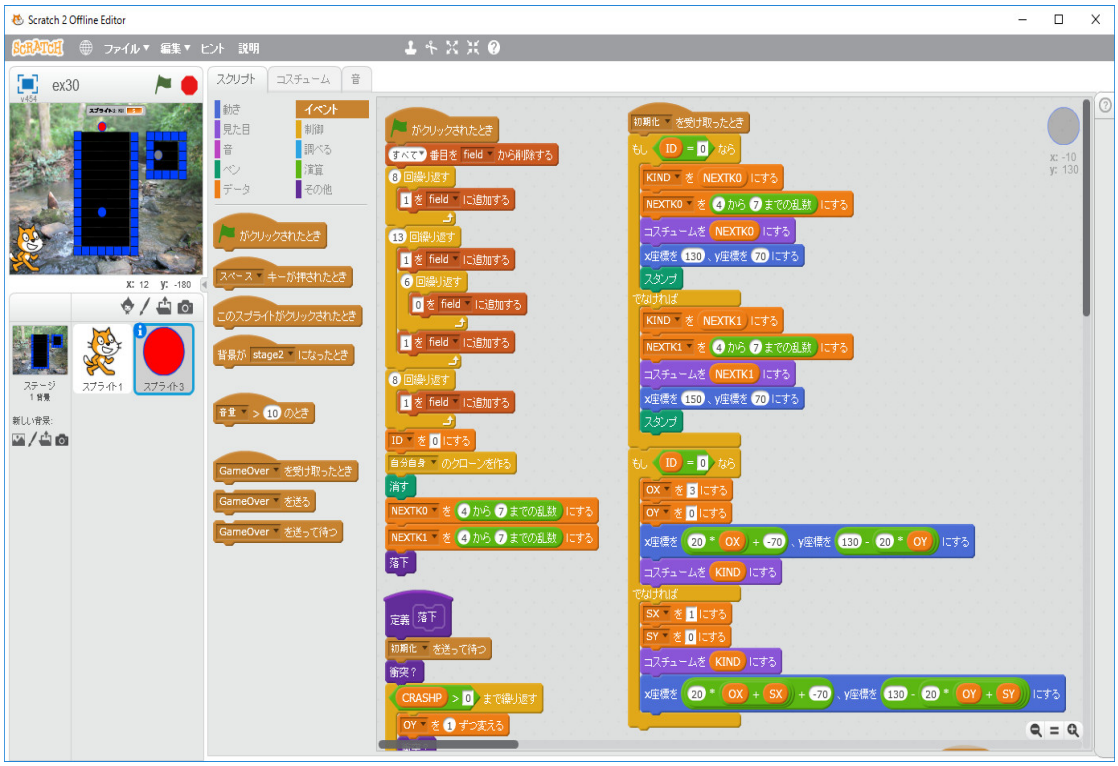
落下した時、めり込まないように表示を修正する。



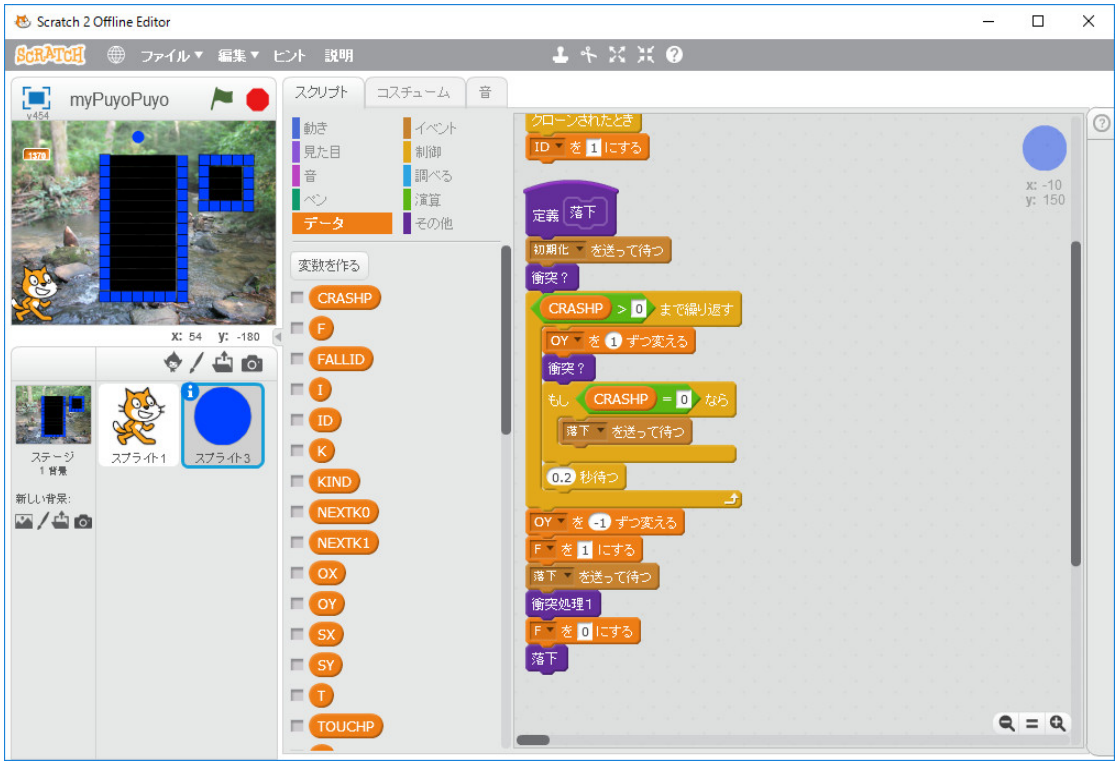
次に、「上向き矢印キーを押したとき」に回転するようにする。「上向き矢印キーを押したとき」の定義は下の図のようにすれば良い。



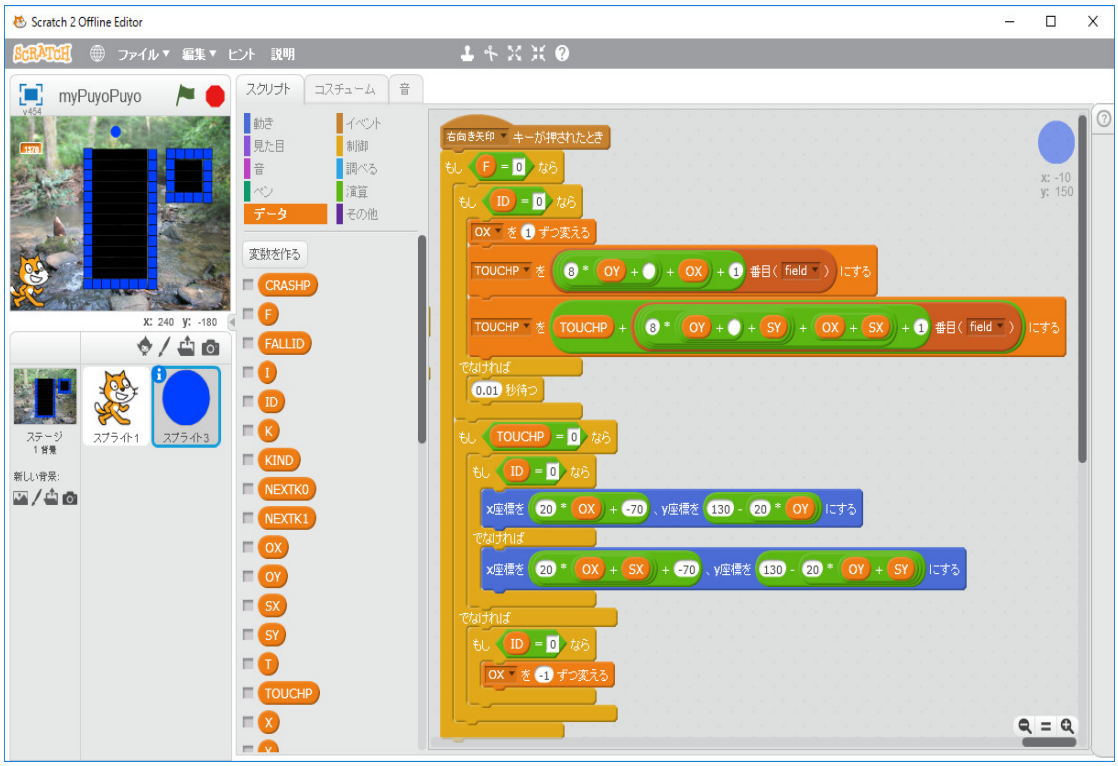
次に、ランダムな組み合わせで「ぷよぷよ」が出てくるようにし、次に出てくる「ぷよぷよ」の予告をするようにする。



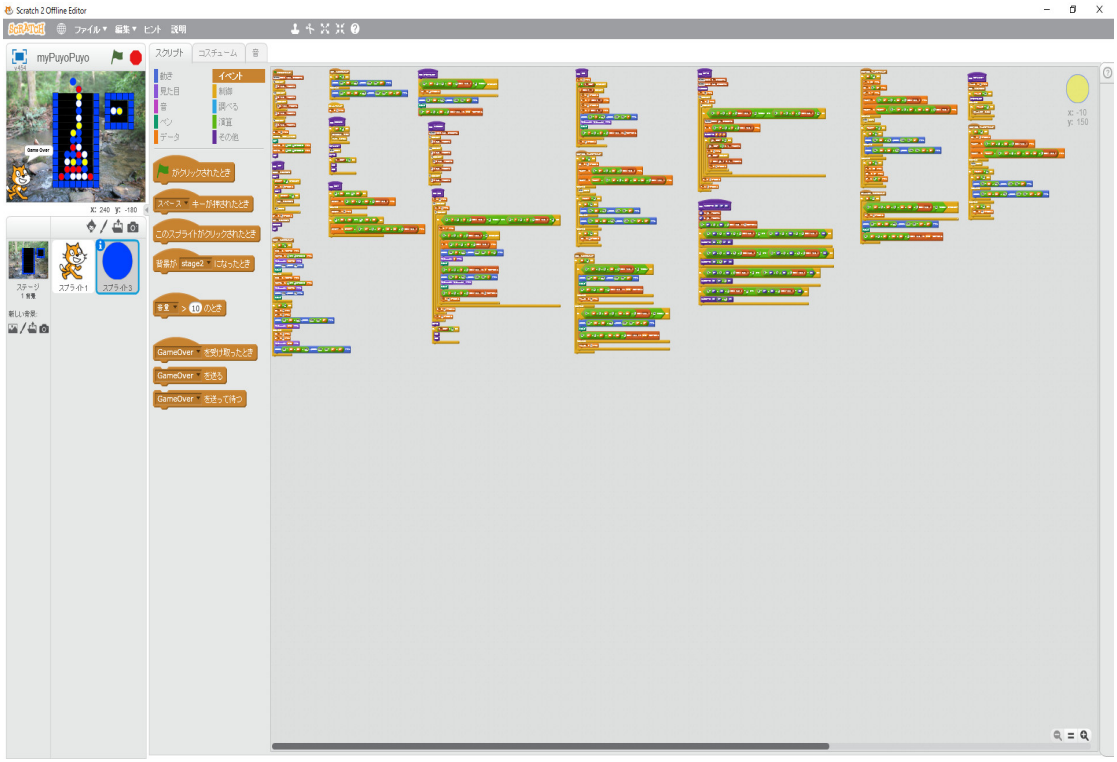
最後に、新たなグローバル変数 F を導入し、「旗がクリックされたとき」に F=0 とセットし、ブロック「落下」で、$CRASHP > 0$ になれば、F=1 とセットし、矢印キーの使用を無効にし、ブロック「衝突処理 1 」が終われば、F=0 とセットし、矢印キーの使用を有効にするように修正します。



「矢印キーが押されたとき」の処理は、「もし F=0 ならば」で全体を囲みます。例えば、「右向き矢印キーが押されたとき」は次のようにします。



プログラムの全体図は次のようになります。



多分、間違っていないと思います。「下向き矢印キーを押したとき」に直ちに落下する機能は作っていません。スコアを計算し、表示する機能も付けていません。必要なら自分で作ってください。

これで、「テトリス」と「ぷよぷよ」の Scratch によるプログラム作成の解説は終わりです。昔 C などで作るのは大変だったゲームのプログラムが Scratch なら簡単にできます。昔遊んだことのあるゲームや新たに自分で考えたゲームを作って楽しんでください。プロのプログラマーになりたい人だけが、プログラミングを学ぶ訳ではありません。商品にはならなくても、趣味で色々なものを作って楽しんでいる人達がいます。例えば、自分のための棚や椅子などの小物を日曜大工で作っている人達や小説やエッセイを書くわけではないが日記を書いたり俳句や川柳を作っている人達が沢山いるように、Scratch でゲームや三角関数を描くプログラムや県名当てパズルのような単純な教材を作ってみて、プログラムを作ることが楽しくなったら、Python や Ruby や JavaScript や C++ といった他の本格的なプログラミング言語にも取り組んでください。きっと、新しい楽しい世界が開けます。知ることは楽しいことです。知らないということは悲しいことです。